UNIVERSITY OF ALBERTA

Neural Processing of Tactile Data

BY

Steven Craig Charlton

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of Master of Science.

DEPARTMENT OF COMPUTING SCIENCE

Edmonton, Alberta
Spring 1994

National Library
of Canada

Acquisitions and
Bibliographic Services Branch

395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale
du Canada

Direction des acquisitions et
des services bibliographiques

395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Your file Votre référence

Our file Notre référence

The author has granted an irrevocable non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of his/her thesis by any means and in any form or format, making this thesis available to interested persons.

L'auteur a accordé une licence irrévocable et non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de sa thèse de quelque manière et sous quelque forme que ce soit pour mettre des exemplaires de cette thèse à la disposition des personnes intéressées.

The author retains ownership of the copyright in his/her thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without his/her permission.

L'auteur conserve la propriété du droit d'auteur qui protège sa thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

ISBN 0-612-11175-X

Canada

UNIVERSITY OF ALBERTA

RELEASE FORM

NAME OF AUTHOR: Steven Craig Charlton

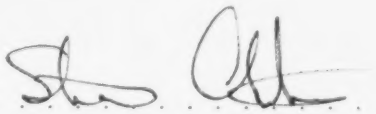
TITLE OF THESIS: Neural Processing of Tactile Data

DEGREE: Master of Science

YEAR THIS DEGREE GRANTED: 1994

Permission is hereby granted to the University of Alberta Library to reproduce single copies of this thesis and to lend or sell such copies for private, scholarly or scientific research purposes only.

The author reserves all other publication and other rights in association with the copyright in the thesis, and except as hereinbefore provided neither the thesis nor any substantial portion thereof may be printed or otherwise reproduced in any material form whatever without the author's prior written permission.

(Signed) 

Steven Craig Charlton
30 Prescot Rd.
Winnipeg, Manitoba
Canada, R3T 3X5

Date: April 21 1994

UNIVERSITY OF ALBERTA

FACULTY OF GRADUATE STUDIES AND RESEARCH

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research for acceptance, a thesis entitled **Neural Processing of Tactile Data** submitted by Steven Craig Charlton in partial fulfillment of the requirements for the degree of Master of Science.

Hong Zhang

Dr. Hong Zhang (Supervisor)

Roger W. Toogood

Dr. R. Toogood (External)

W. W. Armstrong

Dr. W. W. Armstrong (Examiner)

L. Y. Yuan

Dr. L. Y. Yuan (Chair)

Date: April 21 1994

Abstract

Tactile feedback can play an important role in robotic object manipulation and recognition tasks. High level contact parameters such as contact forces and position give information about the state of a manipulation task that cannot be obtained easily from other sources. Unfortunately, these contact parameters are not immediately available from a tactile sensor and must be derived from the sensor's output. In this thesis, artificial neural networks, specifically backpropagation networks, have been used to extract high-level contact parameters from stress profiles. These stress profiles were generated using a Finite Element Analysis model of a force sensing tactile array. The FEA model is free from the inaccuracies introduced by simplifying assumptions made in many analytical models. Data from a physical force sensing array was also used to show the applicability of this approach to the physical problem.

Contents

1 Introduction 1

1.1 Tactile Sensing 2

1.2 Neural Networks 6

1.3 Description of Research 9

1.4 Thesis Layout 9

2 Background 11

2.1 Tactile Sensing 11

2.1.1 Transduction Technology 12

2.1.2 Modeling 13

2.2 Neural Networks 19

2.2.1 Backpropagation 20

2.2.2 Hopfield Networks 27

2.2.3 Adaptive Logic Networks 31

2.3 Previous Applications of Neural Networks to Tactile Sensing 33

3 Experimental Setup **35**

3.1 The Model 35

3.2 The Network 38

3.2.1 Scaling 41

3.2.2 The Simulator 42

3.3 The Experiments 45

3.3.1 Physical Experiment 48

4 Simulation and Experimental Results **49**

4.1 Simulation Results 49

4.1.1 Normal and Tangential Force Recovery 49

4.1.2 Width Recovery 56

4.1.3 Position Recovery 63

4.2 Experimental Results 70

4.3 Summary of Results 70

5 Conclusions and Further Research **74**

5.1 Further Research 75

A Finite Element Model **76**

A.1 Material Properties 76

A.2 FEA Model of Layer 77

A.3 FEA Model of Contact 77

A.4 Modeling Friction 78

Bibliography **81**

List of Figures

1 Depiction of a force sensor array. 4

2 Depiction of the output of one row of force sensing elements when a force is applied at two different angles but the same position. Erroneous estimates of indentor position can be made in the presence of tangential forces. 5

3 The topography of a typical feed–forward neural network. Inputs are fed to a layer of fanout nodes which pass them forward to a layer of processing nodes. These nodes compute weighted sums of their inputs and forward the results to nodes in the the next layer. 7

4 Physical layout of a *Force Sensing Resistor*. 13

5 Stress components of a point in an elastic continuum. 14

6 This graph represents surface stress distributions for three different cases. The solid line is the stress distribution for the analytical model using the half–space assumption. The dotted line represents the FEA model output for a thick layer (1.27 cm). The dashed line represents the FEA model output for a thin layer (0.08 cm). 18

7 Model of a biological neuron 19

8 Topology of a backpropagation network. The boxed area is expanded in Figure 9 to show the connections. 22

9 Depiction of a feedforward connection between input elements and output elements of a processing unit. 23

10 Detailed drawing of processing node i of layer l. 25

11 Output of the sigmoid function $s(I) = 1/(1 + e^{-I})$. 25

12 Topology of a Hopfield recurrent associative network. 28

13 Diagram of the FEA model used to create training and testing data for the neural networks. 36

14 Interface area between indentor and elastic layer in the FEA model. . 37

15 Layout of the backpropagation network. 39

16 SNNS "remote" panel. This panel allows the training and testing of networks, and the adjustment of the various training parameters. . . 42

17 SNNS network activation values display. The output of each processing node in the network is represented in both a numeric and graphical form. 43

18 SNNS Hinton diagram for displaying network weight values. 44

19 Overview of the experiments. 45

20 Stress profiles for different indentor widths and tangential forces. . . . 47

21 Stress profile for varying applied normal and tangential forces. 50

22 Stress profiles for varying normal force with other variables held static. 50

23 Stress profiles for varying tangential force with other variables held static. 51

24 Relative error in normal force. Each graph represents the error in normal force recovery for a given applied normal force. The force values for this figure range evenly from 4.5 N in the top left corner to 13.3 N at the bottom. 52

25 Relative error in normal force. Each graph represents the error in normal force recovery for a given applied normal force. The force values for this figure range evenly from 15.6 N in the top left corner to 21.5 N at the bottom. 53

26 Relative error in tangential force. Each graph represents the error in tangential force recovery for a given applied normal force. The force values for this figure range evenly from 0.45 N in the top left corner to 2.2 N at the bottom. 54

27 Hinton diagram of the trained weight values for the network recovering applied normal and tangential forces only. 55

28 Stress profiles for varying indentor widths with the other variables held constant. 56

29 Relative error in normal force. Each graph represents the error in normal force recovery for a given applied normal force. The force values for this figure range evenly from 4.5 N in the top left corner to 13.3 N at the bottom. 58

30 Relative error in normal force. Each graph represents the error in normal force recovery for a given applied normal force. The force values for this figure range evenly from 15.6 N in the top left corner to 21.5 N at the bottom. 59

31 Relative error in tangential force. Each graph represents the error in tangential force recovery for a given applied normal force. The force values for this figure range evenly from 0.45 N in the top left corner to 2.2 N at the bottom. 60

32 Relative error in indentor width. Each graph represents the error in indentor width recovery for a given width. The widths for this figure range evenly from 0.8 cm in the top left corner to 1.1 cm at the bottom. 61

33 Relative error in indentor width. Each graph represents the error in indentor width recovery for a given width. The widths for this figure range evenly from 1.2 cm in the top left corner to 1.5 cm at the bottom. 62

34 Hinton diagram of the trained weight values for the network recovering contact forces and indentor widths. 63

35 Stress profiles for varying indentor position with other variables held static. 64

36 Relative error in normal force. Each graph represents the error in tangential force recovery for a given applied normal force. The force values for this figure range evenly from 0.45 N in the top left corner to 2.2 N at the bottom. 65

37 Relative error in tangential force. Each graph represents the error in tangential force recovery for a given applied normal force. The force values for this figure range evenly from 0.45 N in the top left corner to 2.2 N at the bottom. 66

38 Absolute error in indentor position. Each graph represents the error in indentor position recovery for a given position. The positions (relative to the center of the sensor) for this figure range evenly from -4.0 mm in the top left corner to -0.8 mm at the bottom. 67

39 Absolute error in indentor position. Each graph represents the error in indentor position recovery for a given position. The positions (relative to the center of the sensor) for this figure range evenly from 0.0 mm in the top left corner to 4.0 mm at the bottom. 68

40 Hinton diagram of the trained weight values for the network recovering contact forces and indentor position. 69

41 Sample stress profile from the training set used in the physical experiment. The normal force applied in this case was approximately 13 Newtons. The stress profile (top) was generated by averaging the rows of the actual sensor output (bottom). 71

42 Normal force recovery for the physical sensor experiment. 72

43 *A simplified model of the elastic layer used in the first part of the paper. The actual mesh density used in the analysis was higher than that shown in this figure.* 78

44 *A simplified model used for examining the relationship between the stress distribution and the applied normal force, tangential force, and the indenter width. In the analysis, the mesh density used was higher than that shown in this model.* 79

45 *The stress distributions with and without the friction model. All forces are measured in dynes/cm. (1) $F_t = 2.2e4$ without friction. (2) $F_t = 2.2e4$ with friction. (3) $F_t = 1.5e5$ without friction. (4) $F_t = 1.5e5$ with friction.* 80

List of Tables

2.1 ALN Transfer Functions 31

3.2 Parameters for the backpropagation learning algorithm. 40

3.3 Summary of contact parameters used in the three experiments 46

4.4 Summary of errors for the network recovering normal and tangential forces only (first experiment) 72

4.5 Summary of errors for the network recovering applied forces and indentor width (second experiment) 73

4.6 Summary of errors for the network recovering applied forces and indentor position (third experiment) 73

4.7 Summary of errors for the network in the physical experiment 73

Chapter 1

Introduction

Dextrous object manipulation is an area of robotics that is receiving much attention in current research. The use of multi-fingered hands for grasping and object recognition is desirable in robotic manipulation as it is based on an intuitive model. Performing fine manipulation requires accurate knowledge of the position of the object to be manipulated. As this knowledge is not always present a priori, determining the object's position with respect to the manipulator at the time of contact may be necessary.

The type of information required to accomplish these goals is most easily obtained through the use of tactile sensors. Contact information obtained from these sensors can make tasks that are otherwise difficult, much easier to accomplish. For example, trying to tie one's shoelaces while wearing a heavy pair of gloves would not be easy. The inability to feel where the lace was on your fingers, or to know whether you had gripped the lace at all, would make the task very difficult. Using tactile sensors to obtain contact information gives increased feedback on the state of the manipulation task and allows a greater degree of control of the manipulation task.

Unfortunately, while tactile sensors can give us some information about the object with which a robot is in contact, this information is generally not in a form that is

directly usable for the control of manipulation tasks. Some processing of the tactile sensor data must be done in order to obtain useful parameters.

1.1 Tactile Sensing

Tactile sense is a very broad term, meaning sensing through direct contact, and many different physical properties can be sensed through such direct contact. Nicholls and Lee [NL89] divided tactile sensors into five groups based on the types of data obtainable from them:

- *Simple Contact* — The sensor can only detect if contact has occurred.
- *Magnitude of Force* — A numerical value representing the amount of force applied to the sensor is returned. This force may be torque, shear, normal force, etc.
- *Three-Dimensional Shape* — Numerical quantification by which the shape of the object facing the sensor can be defined; for example, the linear displacement of an array of probes.
- *Slip* — Determines if the object in contact is moving relative to the sensor in the same plane.
- *Thermal Properties* — The temperature and/or thermal conductivity of the contacted object can be obtained.

While all of these types of sensors are very useful, a large amount of research has been devoted to the use of *magnitude of force* tactile sensor arrays and the interpretation of the data they produce. From this data it is possible to determine not only applied force, but also the shape and position of the indenting object. This makes a

force sensor array a very versatile tool for obtaining tactile sense. This thesis will be concerned with this type of sensor and the data it produces.

The very simplest form of force sensing would be a contact switch at the end of a manipulator. If the manipulator were in contact with an object, the switch would be activated; if no object were in contact, it would not. This form of touch sensing can be useful for very rough manipulation tasks where detecting any contact with the object is sufficient, but lacks precision for use in fine manipulation (for example, pushing a cart vs. grasping an egg). A slight improvement in this model would be to allow the sensor to discriminate between different normal contact forces, or simply to determine how hard the manipulator is pushing on an object (or the object on the manipulator). This incurs more discrimination, and when several of these sensors are placed closely together in an array, they form a *tactile array*. This tactile array can sense different pressures at different points on its surface, and can be used to determine shape information about the object. Although the ability to detect other force components with each sensing element would greatly improve our ability to determine contact information from the sensor output, this type of sensor is not commonly available. In contrast, tactile sensors in which each sensing element measures only normal force are in relative abundance.

Almost all tactile sensors are fairly delicate, and must be covered in a protective layer often made of some type of compliant material such as rubber. This covering layer also provides a firmer contact and improves grip. As well as protecting the sensor, this layer tends to act as a filter, spreading a force applied at one point to adjacent points as well. Unfortunately, with the force being spread across an area, it is difficult to resolve the actual applied force at any point on the sensor surface. This can be overcome by simple integration in the case of a point contact, but contact with complex shapes is not so easily resolved as a force is being applied at more than one point.

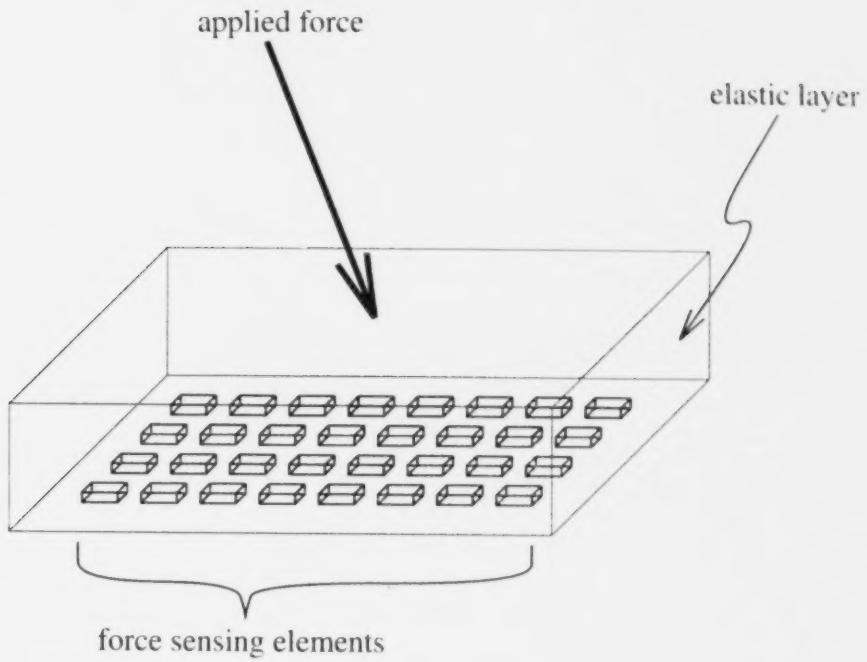


Figure 1: Depiction of a force sensor array.

While the tactile image (consisting of an array of forces) gives us an idea of what a robot is touching, it does not give a complete picture. The convolution of the force data (through the rubber filtering) blurs the tactile image of the object. Thus some sort of deconvolution must be performed on the sensor outputs in order to correctly interpret the data.

When objects are encountered, resistance is not always in a direct line with the direction of motion. It is very useful to know at what angle a force is being applied, information that is not given directly by a normal force sensor array. While this information can be obtained through the use of other sensors (such as shear sensors), it is not always easy to combine them with force sensors. A preferable method is to obtain this information by analyzing the force sensor output.

While a simple tactile sensor can give information as to where on a manipulator a force is being sensed, this information may not accurately show what force is being applied. If the object is applying both a tangential force and a simple normal force

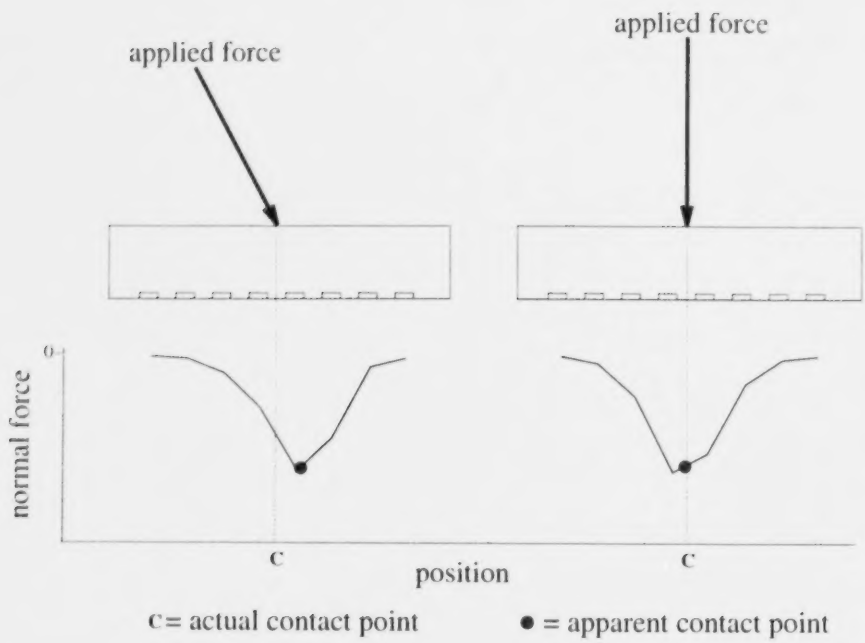


Figure 2: Depiction of the output of one row of force sensing elements when a force is applied at two different angles but the same position. Erroneous estimates of indentor position can be made in the presence of tangential forces.

to the manipulator, the apparent position of contact (determined from normal force readings) could be different from the actual position. This is especially true if simple position determining schemes are used, such as force thresholding (Figure 2).

Removing the filtering effects of the covering layer is a non-trivial task. Several attempts have been made to recover contact information or simply non-filtered surface strain through various methods [TWR92, WS92, PKC+88]. Mathematical deconvolution methods appear to depend on a number of simplifying assumptions about the sensor. Assumptions such as the sensor elements being contained in an elastic half-space[DRCBC91] make mathematical inversion of the convolution problem much easier, but are far from the actual situation in most tactile sensors since most sensors are covered in a thin elastic material and have a rigid backing. The half-space assumption tends to give inaccuracies when the indenting object is wider than the thickness of the elastic layer, or when an object indents near the edge of the

sensor. Without this assumption, closed-form mathematical solutions are difficult to achieve [AK69].

Given the difficulty in using mathematical models, other methods for modeling and interpreting the sensor outputs must be considered. Neural networks have been used in several ways to solve the inversion problem with varying degrees of success [PKP92, TWR92, WS92].

1.2 Neural Networks

Artificial neural networks, while originally an attempt at modeling a biological brain, are now more often used as constructs that allow non-programmed processing of data. These constructs generally take a set of inputs, process them by means of a weight vector, and produce a set of outputs. The weight vector can either be static, or can change in accordance with a learning algorithm. In the latter case, an input is presented to the network, and the difference between the actual output and the desired output is used to adjust the internal weights so that when the same input is presented again, the output is closer to the desired output. This process is repeated on a large set of data to train the network to perform a function. The trained network performs a consistent mapping from the input data to the output data. However, this mapping is generally only an approximation of the function that is needed.

One popular type of neural network is the *feed-forward* network (Figure 3). These networks are organized in a hierarchical manner, with layers of processing nodes passing information forward through the hierarchy. Each layer is, typically, fully interconnected with the next layer forward, allowing each node in a layer to see the output of every node in the previous layer. Each node performs a weighted sum of all its inputs, and then forwards a value based on this sum to the next layer. The value sent forward is calculated according to some type of activation function, which can be different for each node, but is usually a simple sigmoid function. Detailed discussion

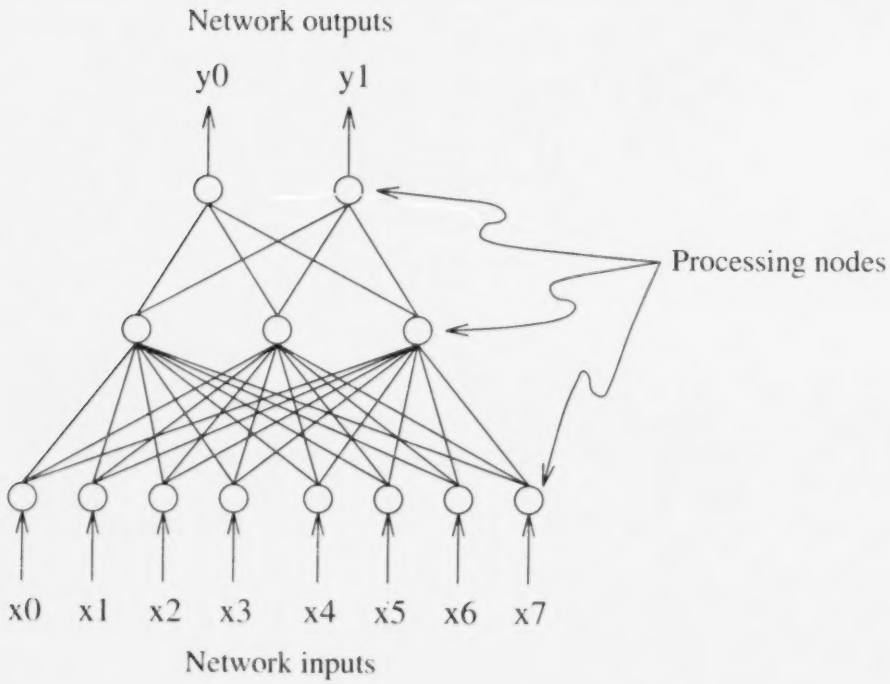


Figure 3: The topography of a typical feed-forward neural network. Inputs are fed to a layer of fanout nodes which pass them forward to a layer of processing nodes. These nodes compute weighted sums of their inputs and forward the results to nodes in the the next layer.

of feed–forward neural networks is postponed until Chapter 2.

While feed–forward networks have been proven able to approximate large classes of functions (including all continuous functions), the accuracy of the network depends on the amount of training and the size of the network (number of processing nodes). With few exceptions, when a neural network is trained on more data and for longer training periods, the accuracy of the function approximation increases. This increase in accuracy is bounded by the size of the network, so that increased network size may be needed to achieve an acceptable approximation. By choosing a correct network size and training on enough data, almost any function can be approximated to almost any accuracy.

Neural networks can be trained on the data directly from the sensors, and therefore it is possible for them to interpret the data without any simplifying assumptions about how the sensor is constructed or how it works. The networks can be created and trained to nearly any accuracy and are often much faster to evaluate than mathematical models, due to their parallel nature. Unfortunately, to obtain high degrees of accuracy, the networks may have to be very large and training time can increase almost exponentially with network size. For this reason, smaller networks are generally used and trained to an acceptable error level.

As mentioned in the previous section, correct mathematical models of tactile sensors are difficult to achieve. In light of this fact, other methods of modeling have been tried. One successful attempt by Pavan Sikka [SZT93] uses a *Finite Element Analysis* (FEA) model that models a sensor with fewer simplifying assumptions than analytical models. This FEA model produces excellent results, but as FEAs are numerical in nature, the model is difficult to invert analytically. Neural networks learn complex non–linear functions by example, and are ideal for inverting models that produce numerical results. The FEA can be used to produce a set of data points that correspond to different contact situations and their resulting sensor outputs. These data points can then be used to train a neural network to invert the tactile sensor

data and to recover contact parameters.

1.3 Description of Research

This thesis describes a set of experiments conducted to determine if artificial neural networks could be used successfully to recover contact information from the output of a tactile sensor. The contact information to be recovered included:

- total normal and tangential forces applied by the indentor
- indentor width
- indentor position
- indentor shape

The majority of the experiments were conducted using tactile images produced by a finite element model of a typical normal force sensor. This model was constructed without the use of the half-space assumption, and is considered to be a reasonably accurate model. Further experiments on data from a physical tactile sensor were conducted to show that the results apply in the physical case as well.

From the results of experiments carried out for this thesis, it will be shown that three-layer backpropagation neural networks can be used to obtain accurate contact information for the typical contact situations of point, line and rectangular contact.

1.4 Thesis Layout

This thesis will be presented in the following manner.

In Chapter 2, a brief introduction to the areas of tactile sensing and neural networks will be presented. A brief survey of literature in the tactile sensing field will provide sufficient background for the reader to understand the application of this thesis to the field. Neural networks will be described in general terms, along with a detailed description of a few different types of networks and their applications. This will be followed by a literature review on neural networks as applied to tactile sensing.

Chapter 3 will describe a solution to the inverse tactile sensing problem. The network used will be described in detail, along with the experiments that were carried out to test the feasibility of using neural network technology to solve the problem. A brief description of the FEA model used in the experiments will be given, while a more complete accounting will be given in the Appendix.

The results of the experiments will be given in Chapter 4. A detailed analysis of these results and an analysis of the trained neural networks will also be presented. Chapter 5 will contain a brief review of the results and a discussion of their relevance and degree of success. In addition, a critical review of the experiments and a discussion of possible future work will be given.

Chapter 2

Background

Tactile sensing is a relatively new field in robotics and has been the target of a large amount of research and development in the recent past. This chapter deals with some of the background information needed to understand the problems involved in the field. First, a review of tactile sensing will be presented with descriptions of the technology and theory involved. Basic neurocomputing concepts will be covered next, with descriptions of three main types of neural networks and their applicability to the problem being addressed in this thesis. Third will follow a review of previous applications of neural networks to the field of tactile sensing and how their approaches differ from that taken in this thesis.

2.1 Tactile Sensing

Human skin has amazing sensing ability. Different nerve endings embedded at different depths in the skin can sense many environmental quantities. Heat, pain and touch are all sensed by nerves that send impulses through the nervous system to the brain. Tactile sensors are an attempt to allow machines the same sort of tactile feedback that we have, making any sort of manipulation task easier.

In chapter one, five different "tactile quantities" were listed as the types of measurements tactile sensors can make. They were: simple contact, magnitude of force, three-dimensional shape, slip and thermal properties. In this thesis, only the force sensing is considered, as it provides information most relevant to manipulation tasks.

2.1.1 Transduction Technology

In the realm of force magnitude sensing there are many different transduction methods that can be used to measure incident force (for a comprehensive description of these methods, see [NL89]). Properties such as conductivity and capacitance are used to design tactile sensors, and each method has advantages and disadvantages.

One simple and popular solution is to use a conductive rubber material between two electrodes. One electrode is fixed against a rigid backing while the other electrode forms the surface of the sensor. As the rubber is compressed, the resistance across the electrodes drops, allowing the force applied to be calculated. This method can also be implemented using carbon fiber mats, and measuring the resistance at the junctions of the perpendicular carbon fibers. As the fibers are pressed together, the contact area increases, creating a decrease in resistance.

A second solution is to use a piezoelectric substance to detect changes in force. Piezoelectric materials generate a voltage when pressure is applied to them. This voltage is proportionally related to the amount of force applied. While piezoelectric polymers can be made with good mechanical properties and durabilities, it can be difficult to get accurate readings from a sensor made out of them. Temperature will often change the effect that pressure has on the voltage output, making accurate readings difficult. Also, the sensors can experience signal decay with constant applied loads.

Capacitance is used in tactile force sensing by placing an elastic dielectric between two parallel electrodes. When a force is applied, the electrodes are pushed closer

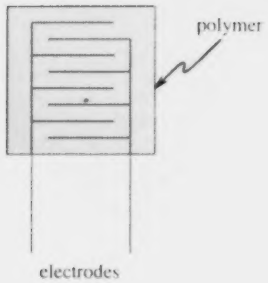


Figure 4: Physical layout of a *Force Sensing Resistor*.

together causing increased capacitance. These sensors are generally constructed as two sets of parallel conductive strips, with a dielectric in between, placed so as to form a grid with the intersections becoming capacitors. The sensor can then be scanned to measure the capacitance at each intersection. There is a limit to how small the surface area of the intersections can be, as capacitance decreases with size and very small capacitance changes can be lost in noise.

Magnetoresistive materials can also be used to measure force. If a current-carrying wire is pressed into a rubber layer so as to bring it closer to a magnetoresistive element, the magnetic field around the wire will cause a measurable change in the resistance of the element. Magnetoelastic materials change their magnetic fields when a force is applied to them. This change can be measured to determine the applied force.

A newer solution by *Interlink Electronics* uses a polymer film whose resistance drops in proportion to an applied force. This film is placed atop a network of electrode "fingers" (Figure 4) and the resistance between the electrodes is measured. Several small patches of this material arranged in a grid-like fashion can be used to sense force applied over an area. Although each element must have a direct connection off the sensor to be read, this solution has excellent sensitivity, range and repeatability [Int89].

2.1.2 Modeling

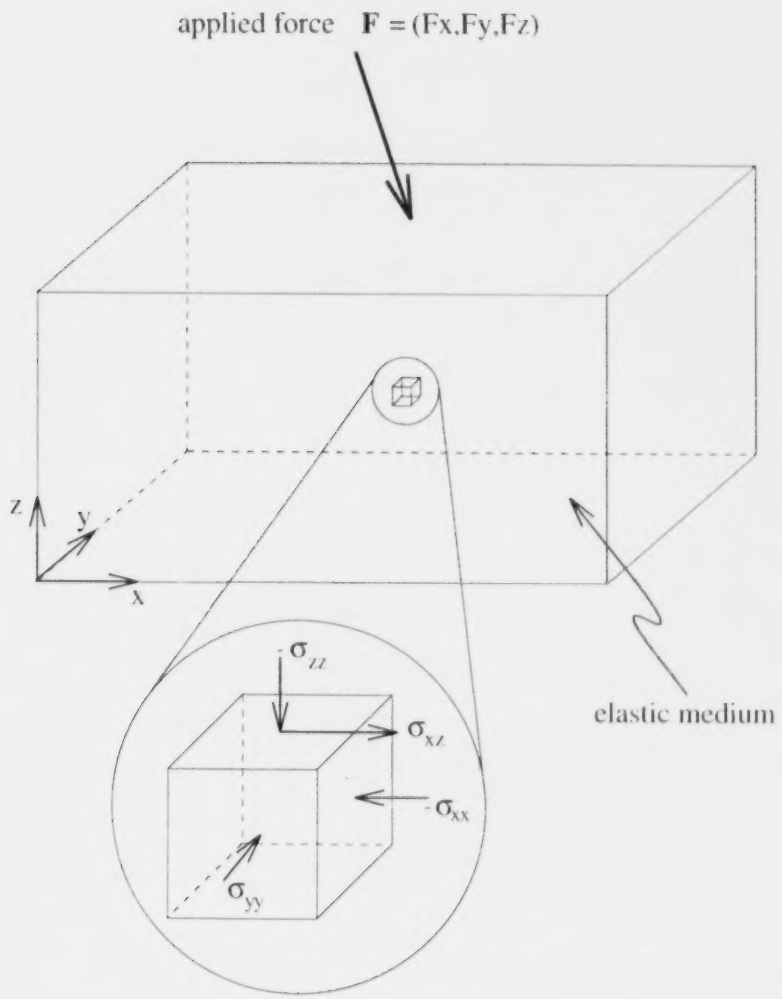


Figure 5: Stress components of a point in an elastic continuum.

The majority of the above mentioned transduction techniques are based on measuring strain as opposed to stress (with the exception of piezoelectric and force sensing resistor methods). Strain (ϵ) is defined as the deformation of a body per unit length

$$\epsilon_x = \frac{\delta u}{\delta x}, \epsilon_y = \frac{\delta v}{\delta y}, \epsilon_z = \frac{\delta w}{\delta z} \tag{2.1}$$

where (u, v, w) is the displacement of a point in an elastic continuum due to an applied stress [Fea92]. Stress (σ) is defined as the applied force per unit area (for example, the force resulting on one side of the infinitesimal cube shown in Figure 5). The relationship between the applied surface force and the subsurface stresses can be expressed as a solution to the following 15 equations[Bur87, BMP65], along with any boundary conditions that apply.

Assuming equilibrium, a system of three partial differential equations can be derived that relate the components of the stress tensor and the external forces acting on the body ((f_x, f_y, f_z), which are generally taken to be negligible).

$$\frac{\partial \sigma_{xx}}{\partial x} + \frac{\partial \sigma_{yx}}{\partial y} + \frac{\partial \sigma_{zx}}{\partial z} = -f_x \tag{2.2}$$

$$\frac{\partial \sigma_{xy}}{\partial x} + \frac{\partial \sigma_{yy}}{\partial y} + \frac{\partial \sigma_{zy}}{\partial z} = -f_y \tag{2.3}$$

$$\frac{\partial \sigma_{xz}}{\partial x} + \frac{\partial \sigma_{yz}}{\partial y} + \frac{\partial \sigma_{zz}}{\partial z} = -f_z \tag{2.4}$$

Assuming linear elasticity and ignoring second-order terms, the strains are expressed as a function of the displacements.

$$\epsilon_{xx} = \frac{\partial u}{\partial x} \tag{2.5}$$

$$\epsilon_{yy} = \frac{\partial v}{\partial y} \tag{2.6}$$

$$\epsilon_{zz} = \frac{\partial w}{\partial z} \tag{2.7}$$

$$\epsilon_{xy} = \frac{1}{2}\left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\right) \tag{2.8}$$

$$\epsilon_{yz} = \frac{1}{2}\left(\frac{\partial v}{\partial z} + \frac{\partial w}{\partial y}\right) \tag{2.9}$$

$$\epsilon_{zx} = \frac{1}{2}\left(\frac{\partial w}{\partial x} + \frac{\partial u}{\partial z}\right) \tag{2.10}$$

A set of six partial differential equations are derived from these functions. These are referred to as the compatibility conditions.

$$\frac{\partial^2 \epsilon_{xx}}{\partial y^2} + \frac{\partial^2 \epsilon_{yy}}{\partial x^2} = 2\frac{\partial^2 \epsilon_{xy}}{\partial x \partial y} \tag{2.11}$$

$$\frac{\partial^2 \epsilon_{yy}}{\partial z^2} + \frac{\partial^2 \epsilon_{zz}}{\partial y^2} = 2\frac{\partial^2 \epsilon_{yz}}{\partial y \partial z} \tag{2.12}$$

$$\frac{\partial^2 \epsilon_{zz}}{\partial x^2} + \frac{\partial^2 \epsilon_{xx}}{\partial z^2} = 2\frac{\partial^2 \epsilon_{zx}}{\partial z \partial x} \tag{2.13}$$

$$\frac{\partial}{\partial x}\left(\frac{\partial \epsilon_{xy}}{\partial z} - \frac{\partial \epsilon_{yz}}{\partial x} + \frac{\partial \epsilon_{zx}}{\partial y}\right) = \frac{\partial^2 \epsilon_{xx}}{\partial y \partial z} \tag{2.14}$$

$$\frac{\partial}{\partial y}\left(\frac{\partial \epsilon_{yz}}{\partial x} - \frac{\partial \epsilon_{zx}}{\partial y} + \frac{\partial \epsilon_{xy}}{\partial z}\right) = \frac{\partial^2 \epsilon_{yy}}{\partial z \partial x} \tag{2.15}$$

$$\frac{\partial}{\partial z}\left(\frac{\partial \epsilon_{zx}}{\partial y} - \frac{\partial \epsilon_{xy}}{\partial z} + \frac{\partial \epsilon_{yz}}{\partial x}\right) = \frac{\partial^2 \epsilon_{zz}}{\partial x \partial y} \tag{2.16}$$

(2.17)

Finally, assuming the medium to be isotropic, a set of six equations can be derived that relate the stress tensor to the strain tensor.

$$\begin{Bmatrix} \sigma_{xx} \\ \sigma_{yy} \\ \sigma_{zz} \\ \sigma_{xy} \\ \sigma_{yz} \\ \sigma_{zx} \end{Bmatrix} = \frac{E}{(1+\nu)(1-2\nu)} \begin{bmatrix} 1-\nu & \nu & \nu & 0 & 0 & 0 \\ \nu & 1-\nu & \nu & 0 & 0 & 0 \\ \nu & \nu & 1-\nu & 0 & 0 & 0 \\ 0 & 0 & 0 & 1-2\nu & 0 & 0 \\ 0 & 0 & 0 & 0 & 1-2\nu & 0 \\ 0 & 0 & 0 & 0 & 0 & 1-2\nu \end{bmatrix} \begin{Bmatrix} \epsilon_{xx} \\ \epsilon_{yy} \\ \epsilon_{zz} \\ \epsilon_{xy} \\ \epsilon_{yz} \\ \epsilon_{zx} \end{Bmatrix} \tag{2.18}$$

Thus, problems assuming linear elasticity and isotropic materials are expressed in terms of the above 15 equations involving the 15 unknowns (6 stress components, 6 strain components, and 3 displacement components). Solving this set of equations, along with any boundary constraints, analytically is hard unless certain assumptions about the model are taken (possibly relaxing some boundary constraints).

The construction of most sensors and sensor models is based upon placing sensing elements at some depth in the elastic medium depicted in figure 5. In the case of the *Interlink* sensor, σ_{zz} is measured at a fixed depth beneath the top of the sensor medium. As mentioned above, the large system of equations is very difficult to solve analytically without the aid of simplifying assumptions, making modeling of tactile sensors hard. The most useful assumption (and the one most often made) is that of an infinite half-space, that is, the elastic medium extends infinitely in two axes (x and y), and conforms to a constraint such as $z < 0$ in the third. Without this assumption, the existance of the stress functions throughout the medium cannot be proved [Pav94].

The half-space assumption does not hold in all cases. Numerical approximation methods can often be used to provide a solution without assumptions. One such method is Finite Element Analysis which uses differential equations to obtain a displacement field in a discretized medium.

Pavan Sikka developed a finite element analysis model of a force sensing tactile sensor [SZT93] which he used to show the invalidity of the half-space assumption for thin sensors. Figure 6 shows the stress distribution at the bottom of elastic layers of different thicknesses as calculated by the FEA model and analytically using the half-space assumption. It is quite clear that the assumption breaks down as the thickness of the layer decreases. For this reason, the finite element analysis method will be used throughout this thesis for the generation of tactile data.

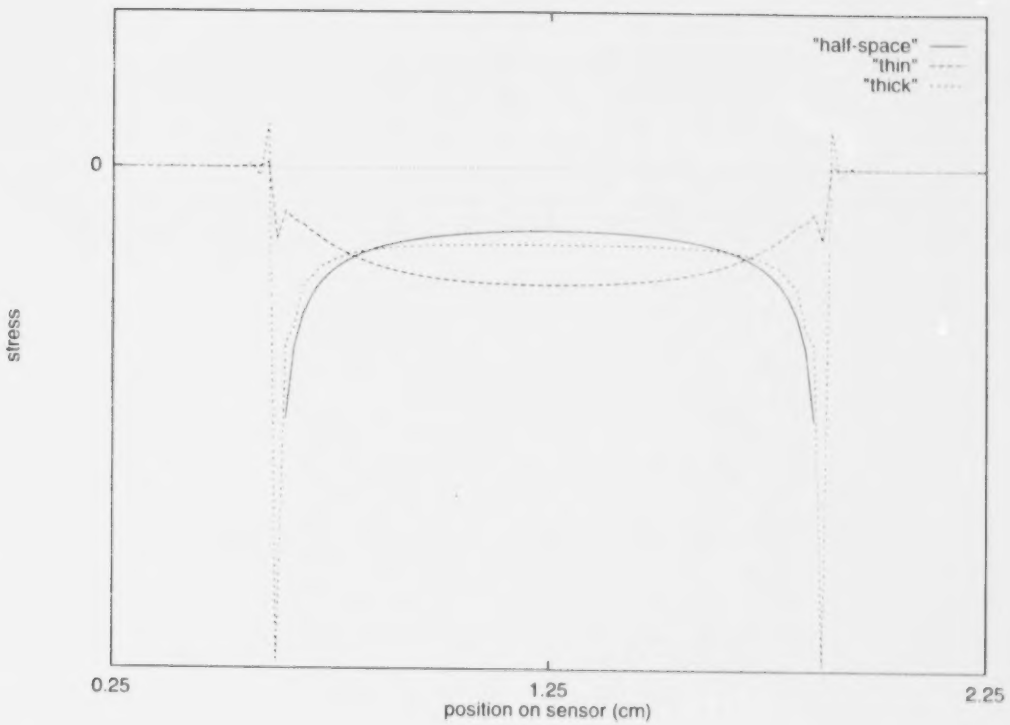


Figure 6: This graph represents surface stress distributions for three different cases. The solid line is the stress distribution for the analytical model using the half–space assumption. The dotted line represents the FEA model output for a thick layer (1.27 cm). The dashed line represents the FEA model output for a thin layer (0.08 cm).

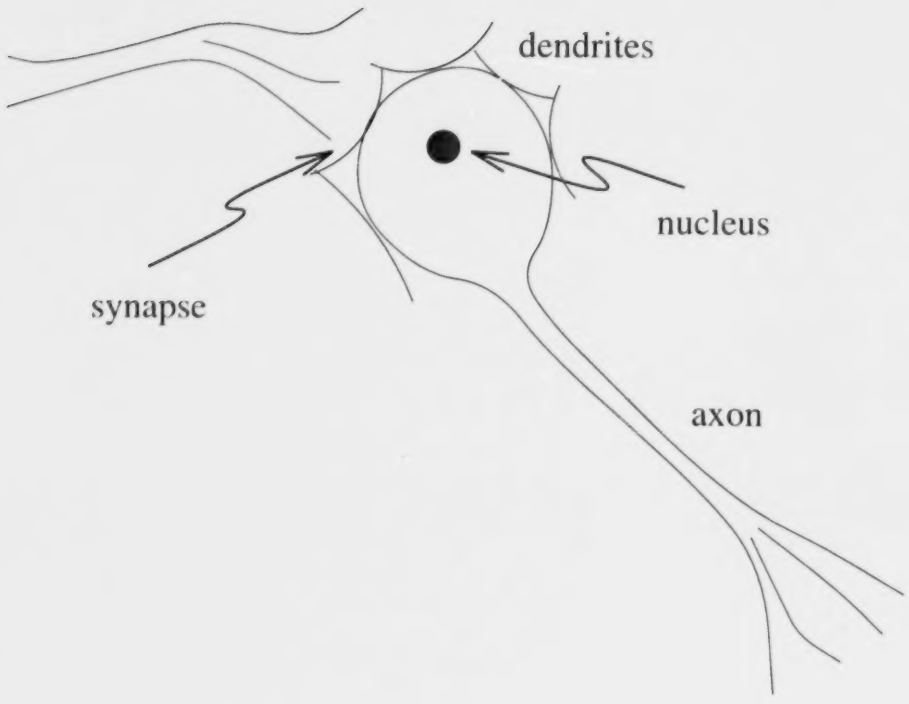


Figure 7: Model of a biological neuron

2.2 Neural Networks

The field of neurocomputing started as an attempt to model biological neural systems. The original hope was that a good model would allow a better understanding of how our own brains work. As it turned out, neural nets are not a very good model of the brain, but they do provide a platform for a type of computing that is quite removed from conventional programming.

Neural networks are made up of individual processing elements (nodes) that act in a manner very similar to biological neurons (see Figure 7). In a neuron, signals from other neurons build up around the nucleus, and if a sufficient level of signal is reached, the neuron fires a signal through the axon toward another neuron. In the artificial network's node, the inputs from other nodes in the network are all weighted according to a weight matrix, and are summed together. This result then has an

activation function applied to it to determine the output of the node. This activation function is often implemented as a simple sigmoid function with range $[-1, 1]$. The end result is that the activation function acts as a threshold function, much like the biological neuron. When a large number of these nodes are connected together, the resulting network is similar (at least topologically) to biological neural networks.

In the late 1950's, Frank Rosenblatt invented the *perceptron* [Ros58], a neural network processing node that behaves in a fashion similar to that described above. This perceptron computed the weighted sum of its inputs, and produced an output of 0 if this sum was negative, or 1 if the result was positive. Rosenblatt and some collegues built a machine that was the first functional neural network, called the *Mark I Perceptron*. It functioned as a character recognition system, classifying 20×20 pixel images into a set of categories corresponding to the letters of the alphabet. While this was a tremendous achievement, the perceptron fell short of the neural network ideal, as it was proved unable to compute certain functions, including exclusive-or.

Despite this, research into neural networks continued, and several different types of networks evolved. This section will describe two of the most popular types of networks.

2.2.1 Backpropagation

A feed forward neural network is one in which the processing nodes (units) are grouped into layers, and communication between layers is strictly one way. The inputs to the network are picked up by the input layer, and propagated to the first processing layer. The outputs of the first processing layer are passed on to the second processing layer and so on, until eventually being passed on to the output layer. The processing layers are also called *hidden* layers, as they have no connections outside of the network. Each processing node (and each output node) performs some transfer function, combining the inputs and producing an output.

In 1957, Andrei Kolmogorov published a critical theorem that led to an increased activity in research into feedforward neural networks. That theorem is stated as follows:

Theorem 1 (Kolmogorov's Mapping Neural Network Existence Theorem) *Given any continuous function* $f: [0,1]^n \rightarrow \mathbf{R}^m$, $f(\mathbf{x}) = \mathbf{y}$, f *can be implemented exactly by a three-layer feedforward neural network having* n *fanout processing elements in the first* ($\mathbf{x}$ *- input) layer,* $(2n+1)$ *processing elements in the middle layer, and* m *processing elements in the top* ($\mathbf{y}$ *- output) layer. The size of the input vector is* n*, and* m *is the size of the output vector.*

This theorem implied that neural networks could be trained to perform nearly any function perfectly. Kolmogorov's theorem was proved several years later, but as the proof was non-constructive, it is not known what transfer function must be performed by the hidden and output layers of the *Kolmogorov mapping network*. The theorem, however, provoked a large amount of effort in developing feedforward networks and this effort eventually produced the *backpropagation network*, one of the most popular neural network architectures in use today.

The backpropagation network is a hierarchical feedforward network that can be used to approximate a function f by training on example points (x_1,y_1), (x_2,y_2), ..., (x_k,y_k), ... where x and y can be vectors and $y_k = f(x_k)$. It is assumed that these examples are chosen randomly according to a fixed probability function $\rho(x)$. The architecture consists of a hierarchical set of fully interconnected layers of processing units. The first layer (the input layer) consists of n fanout elements (one per input) that simply supply each of the processing units of the second layer with all of the inputs. The last layer of the network consists of m units and produces the network's estimate of the function output (Figure 8). There can be any number of hidden layers between the input and output layers. However, as the Theorem 1 states that only three-layer networks are needed, only these will be discussed in this chapter.

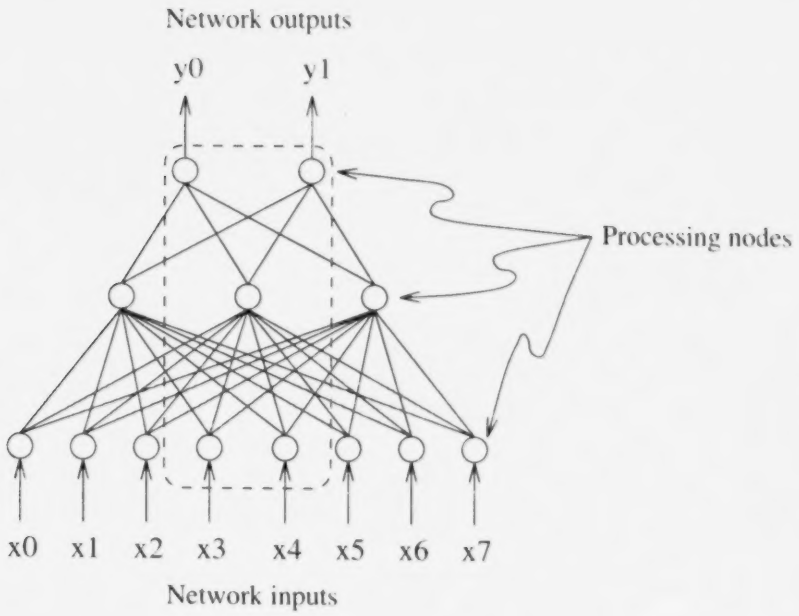


Figure 8: Topology of a backpropagation network. The boxed area is expanded in Figure 9 to show the connections.

Each unit of each layer is connected to every unit of the next layer through the feedforward connections. In addition to these, each unit of each hidden layer receives a feedback connection from each unit in the layer above it (see Figure 9). These feedback connections are used to propagate error signal backward through the network during the learning process (hence the name backpropagation).

Each processing unit in the hidden and output layers can be thought of as a single output processing element and several input processing elements. Each input processing element sends a feedforward signal (generally its input, weighted by some stored value) to its output element and a feedback signal to the output element of the previous layer that provided it with input. The output elements each send feedback signals to all of their input elements, and feedforward signals to one of the input elements of each of the processing units in the next layer. The feedforward signals of the output layer unit's output elements are the y_i components of the network output.

The backpropagation network functions in two stages. First, during the forward

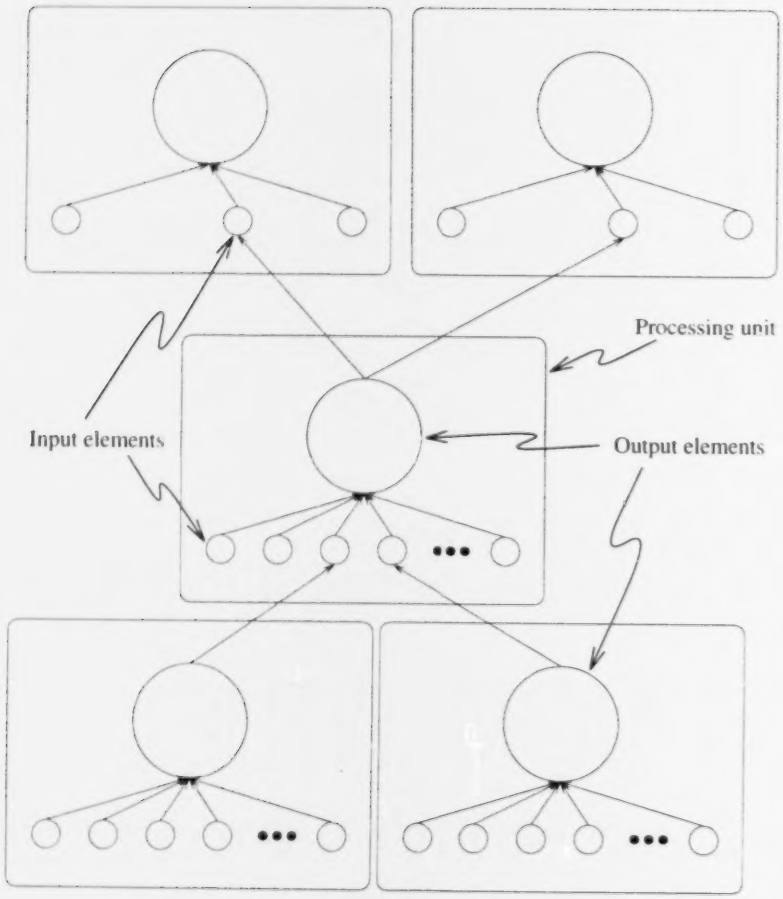


Figure 9: Depiction of a feedfor···ard connection between input elements and output elements of a processing unit.

pass, the input is latched onto the first layer and each layer is activated in turn, with the data propagating through the network until the m output units emit the network's approximation of $f(x_i)$. The correct value is then given to the output elements of the output layer, and the backward pass begins. Each of the output elements computes the error in its output and transmits that error to its input elements who optionally update their weights and transmit the error backward to the previous layer. This continues through the layers until all of the input elements have received an error signal and possibly updated their weights. The cycle can then be repeated with another (or the same) input value. These two passes will be discussed in more detail in the following section.

Backpropagation Algorithm

On the forward pass, each input element in the currently active layer takes the value supplied to it by the previous layer, and multiplies it by the internally stored weight value. The result is then passed on to the output element.

The output element sums the inputs received from the input elements and stores that sum. It then applies a sigmoid function to the sum in order to produce the output that is passed on to the next layer of the network, or (if this is the output layer) out of the network.

$$O_{li} = s\left(\sum_{k=1}^{M_{(l-1)}} w_{lki} O_{(l-1)k}\right) \tag{2.19}$$

where O_{li} is the output for unit i of layer l, s is the sigmoid function $s(I) = 1/(1 + e^{-I})$(Figure 11), $M_{(l-1)}$ is the number of units on layer $(l-1)$, w_{lki} is the weight of input element k of unit i on layer l, and $O_{(l-1)}$ (Figure 10).

During training, this forward pass is followed by a backward pass in which learning (or at least adaptation) occurs. The output elements of the output layer calculate the

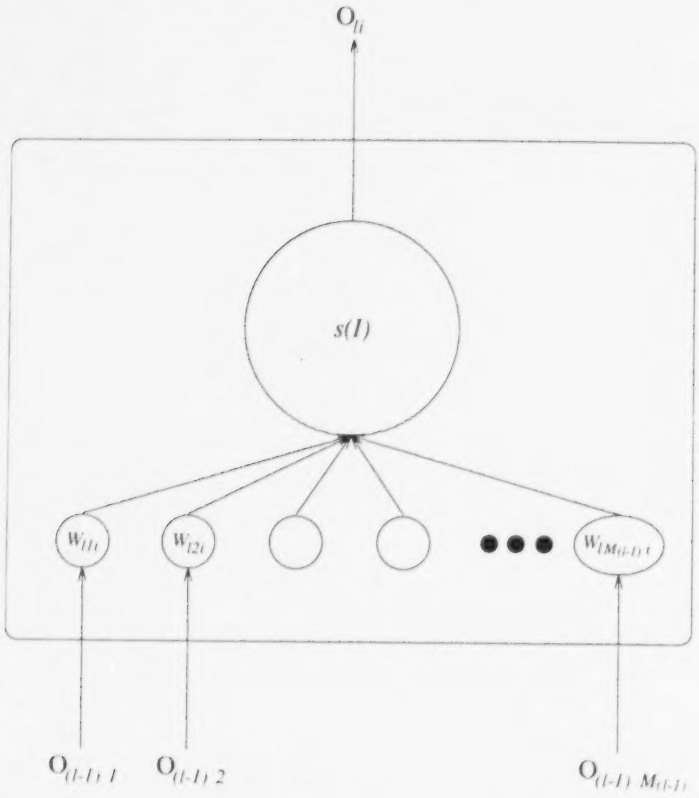


Figure 10: Detailed drawing of processing node i of layer l.

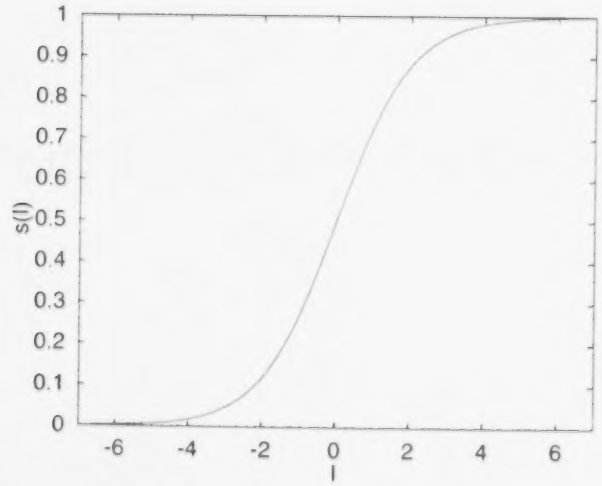


Figure 11: Output of the sigmoid function $s(I) = 1/(1 + e^{-I})$.

difference between their stored output and the desired output supplied for training. This value is then passed to all the input elements where it is multiplied by a constant *learning rate* (usually small, near 0.1). This new error value is then added to the weight for the input element. The unmodified error is multiplied by the original weight, and is passed back to the output element of the previous layer.

The hidden layer output elements receive the error values from the input elements of the layer above them and calculate a new error value as derived in [HN89]:

$$\delta_{li} = s'(I_{li}) \sum_{k=1}^{M_{(l+1)}} w_{(l+1)ki}\delta_{(l+1)k} \tag{2.20}$$

where δ_{li} is the error for output element i of layer l, s is the sigmoid function and $s' = ds/dI$, I_{li} is the sum stored by output element i of layer l during the forward pass, $M_{(l+1)}$ is the number of units on layer $(l+1)$, and $w_{(l+1)ki}$ is the weight of input element i of unit k on layer $(l+1)$. This new δ value is passed back to the unit's input elements which process the error in the same manner as the input elements of the output layer.

The weights are updated as a function of the δ values and the learning rate, α, in the following manner:

$$\Delta w_{lij} = \alpha\delta_{li}O_{(l-1)j} \tag{2.21}$$

where Δw_{lij} is the change in the weight w_{lij}.

Analysis

It has been proved [HN89] that a three layer backpropagation network can approximate, to any desired degree of accuracy, any continuous function and any discontinuous function that is piecewise continuous on a finite number of subsets of $[0,1]^n$.

Although this may take an impractically large number of hidden layer nodes, and must have the correct weight matrix, this result makes backpropagation a very attractive solution.

A problem associated with using backpropagation is that of overtraining. While training the network on a set of training data, it is possible that, in trying to match the training data more exactly, the network will "learn" a different function that fits those points. This can completely ruin the interpolative performance of the network (eg. learning a sine function instead of $f(x) = 0$). Short of testing the network after each training iteration, there is no known way of determining when the network is finished training.

Backpropagation neural networks learn by example the mapping between a set of real-valued inputs to a set of real-valued outputs. If the number of processing units is not very large, they train reasonably quickly (typically in the order of thousands of training iterations) to an acceptable error level. In the absence of overtraining, these networks perform remarkably well in interpolating results although this is not the case with extrapolation.

2.2.2 Hopfield Networks

Unlike feedforward networks, recurrent associative networks have a very strong accretive behavior, mapping similar inputs to the same output. That is, a recurrent associative network would have a mapping function v of the following form

$$v(x_i + \epsilon) = y_i$$

(where ϵ is a sufficiently small error vector), where as a feedforward associative network (such as a backpropagation network) would have a mapping function of

$$v(x_i + \epsilon) = y_i + \delta$$

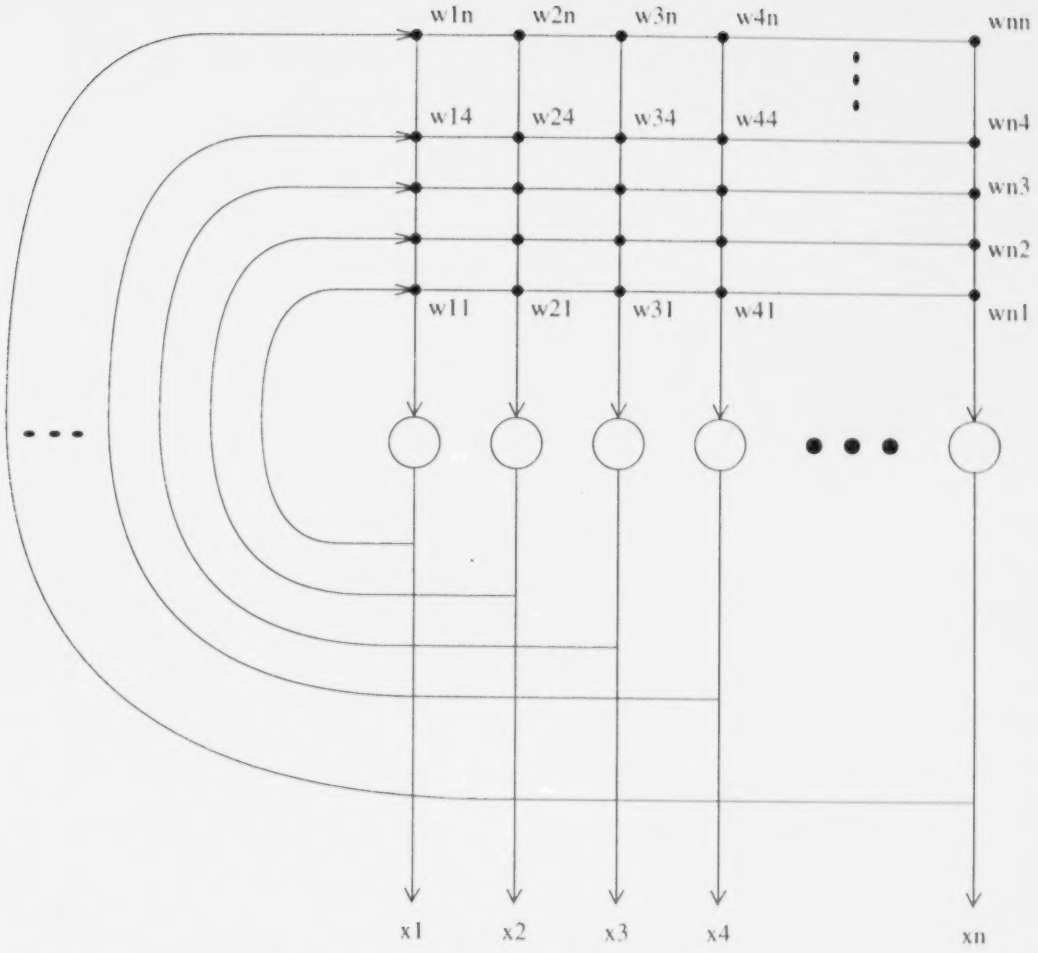


Figure 12: Topology of a Hopfield recurrent associative network.

where $|\delta| \to 0$ as $|\epsilon| \to 0$.

Recurrent associative networks are generally started at some initial state corresponding to the input vector, and then converge to one of a finite number of stable states. The state converged on should be the closest stable state to the initial state, as measured by some metric.

In a 1982 paper [Hop82], John Hopfield described a recurrent associative network that later became known as the *Hopfield network*. This network consists of n processing elements, where n is the number of inputs to the network. Each processing

element receives as input the weighted sum of all the inputs (whose values may be either 1 or -1). A separate set of weight values is associated with each processing element. The outputs of the processing elements are connected back to the inputs (see Figure 12). The processing elements each perform the transfer function

$$x_i^{new} = \begin{cases} 1 & \text{if } \sum_{j=1}^{n} w_{ij} x_j^{old} > T_i, \\ x_i^{old} & \text{if } \sum_{j=1}^{n} w_{ij} x_j^{old} = T_i, \\ -1 & \text{if } \sum_{j=1}^{n} w_{ij} x_j^{old} < T_i. \end{cases} \tag{2.22}$$

for $i = 1, 2, \ldots, n$ and T_i is a threshold value. The processing elements are updated one at a time, usually in a random order, such that all elements update with the same average rate.

As the $n \times n$ weight matrix must be fixed, no learning is associated with the Hopfield network. Instead, the weight matrix is specified in advance, and determines (along with the threshold values T_i) the stable states of the network. In order to guarantee that the network will converge in a finite number of updates, the weight matrix must be symmetric and have a 0 diagonal.

The stabilization behavior of the Hopfield network can be characterized by doing an analysis of the energy function associated with Hopfield networks [Hop82]:

$$H(x) = -\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij} x_i x_j + 2\sum_{i=1}^{n} T_i x_i.$$

This function always decreases if any of the processing elements outputs (x_i) change. For example, assume that processing element k ($1 \leq k \leq n$) has just changed state. The change in energy is

$$\begin{aligned} \Delta H &= H(x^{new}) - H(x^{old}) \\ &= -\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij} x_i^{new} x_j^{new} + 2\sum_{i=1}^{n} T_i x_i^{new} \end{aligned}$$

$$+\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}x_i^{old}x_j^{old} - 2\sum_{i=1}^{n} T_i x_i^{old}$$

$$= -2x_k^{new}\sum_{j=1}^{n} w_{kj}x_j^{new} + 2T_k x_k^{new} + 2x_k^{old}\sum_{j=1}^{n} w_{kj}x_j^{old} - 2T_k x_k^{old} \tag{2.23}$$

Since $x_i^{new} = x_i^{old}$ for $i \neq k$, the terms that do not contain x_k cancel each other out and $w_{ii} = 0$, we get equation 2.23. By factoring out $(x_k^{old} - x_k^{new})$ we can obtain

$$\Delta H = 2(x_k^{old} - x_k^{new})[\sum_{j=1}^{n} w_{kj}x_j^{old} - T_k]. \tag{2.24}$$

From equation 2.22, if x_k has changed from -1 to 1, then $\sum_{j=1}^{n} w_{kj}x_j^{old} - T_k$ must have been positive, and since $x_k^{old} - x_k^{new} = -2$, ΔH will be negative. Similarly, if x_k has changed from 1 to -1, then $\sum_{j=1}^{n} w_{kj}x_j^{old} - T_k$ must have been negative, and since $x_k^{old} - x_k^{new} = +2$, ΔH will again be negative.

From these results, it can be shown that the energy function H will reach a minimum value in a finite number of steps, and that these minima all lie at local minima. Unfortunately, the stable state reached may not be the closest stable state to the input state, as the order of updates of the processing elements affects the change of state of the network.

Analysis

As mentioned above, a Hopfield network will always stabilize in a finite number of steps, but this stable state will not always be the stable state that lies closest to the initial state. The settling time of such a network could also be very large, as the total number of states is 2^n, and it is conceivable that the network might visit a large number of these states during descent to minimum energy.

As the Hopfield network is non-adaptive, the weight matrix must be specified in

Function	*Description*
left	pass the left input directly
right	pass the right input directly
and	pass the logical **and** of the two inputs
or	pass the logical **or** of the two inputs

Table 2.1: ALN Transfer Functions

advance. It is a non-trivial task to find a weight matrix that will yield an acceptable number of stable states at reasonably distributed points in the input vector space.

Accretive networks essentially perform the task of taking input vectors and classifying them into discrete groups where similar inputs are grouped together. While this kind of network can be used to learn functional mappings, it requires much larger networks than would be needed of an interpolative network. As this is the case, this type of network is not suited to the problem being addressed in this thesis.

2.2.3 Adaptive Logic Networks

Another form of feedforward network is the *Adaptive Logic Network* developed by W. W. Armstrong. In this network, only boolean values are used as inputs and passed between processing elements. The architecture of an adaptive logic network (or ALN) is that of a set of binary trees of processing elements, of which each tree produces one bit of the output vector. In every tree, each of the processing elements computes a boolean logic function of its two inputs and passes the result to a processing element in the level above it. There are four different transfer functions (Table 2.1) that can be performed by a processing element, and learning is accomplished by changing the transfer function of one (or more) of the processing elements.

In training, the output of the ALN tree is compared to the desired output. If the values differ, the learning algorithm recurses through the tree discouraging the behavior. If the values are the same, the behavior is reinforced. Processing elements

that have inputs of (0.0) or (1.1) have no effect on the output of the network, as for these inputs, all the transfer functions produce the same output. These nodes are not responsible for the error (or correct answer). Those elements that have inputs of (0.1) or (1.0) should be adjusted according to the output error. Each processing element has two counters associated with it that keep track of the number of times a (0.1) or (1.0) value is encountered. If the desired output is a 1, then the appropriate counter is incremented. If the desired output is a 0, then the counter is decremented. The counters are bounded and can only reach values of a certain absolute magnitude, after which they can only be moved toward 0.

Analysis

Adaptive logic networks, taking boolean-valued inputs, are difficult to use when real-valued inputs and outputs are required. These values must be encoded into binary vectors in such a way that the vectors used to represent two similar values differ in very few bits. Few coding schemes are able to do this while maintaining a large number of code-words in a small number of bits. This means that a very large number of boolean inputs must be used to represent a real-valued number to some reasonable precision. Large trees must be used to allow the inputs to be represented and, since each tree produces only one bit of output, many of these trees must be trained in order to get a real-valued output. The result is that, while ALNs interpolate well and have a good immunity to noise, the sizes of the networks can become prohibitive.

To reduce the number of trees required, the network can be trained to determine whether the desired result is higher or lower than a supplied estimate of the result. Thus a single tree can be used to perform binary searches for the correct result. While this is an improvement in network size, training time will increase dramatically as the network must be shown several examples of values that are either under or over the desired result. These problems make ALNs an unsatisfactory network topology for solving the problem described in this thesis.

2.3 Previous Applications of Neural Networks to Tactile Sensing

In previous studies, the use of artificial neural networks (ANNs) with respect to the problems encountered in tactile sensing has been confined to solving two basic problems: recovering contact surface stress, and recovering indentor shape. In the literature surveyed, none of the studies made use of data from real tactile sensors in training or testing the neural networks. Furthermore, none of these studies attempted to extract the contact forces as a higher level feature available from tactile sensors.

DeRossi *et al.* [CMDR+92] used backpropagation neural networks to recover the radius of contact and the indenter shape from the normal and shear stress profiles. The training data was obtained by solving a contact model based upon a number of limiting assumptions such as an axi-symmetric indenter contacting an elastic half space. As was shown earlier in this chapter, the use of the half-space assumption can produce incorrect data causing improper network training. Also, shear stresses are supplied by very few tactile sensors, limiting the usefulness of this research.

The use of backpropagation neural networks to perform feature extraction and classification of two-dimensional tactile images was shown by Thint *et al.* [TWR92]. The network would classify any tactile image into one of nine classifications. These classifications corresponded to the nine shapes used in training the network. Appreciable degradation in performance was observed when noise was added to the tactile images.

Worth and Spencer [WS92] used a backpropagation neural network to determine the contact angle of a rod impinging on a cylindrical finger. The addition of force and indenter radius variations in the data caused the output quality to degrade slightly.

Another work [PKC+88] used a maximum entropy deconvolution network similar to a Hopfield network to recover contact surface stress distribution from the strain

profile. As mentioned in the previous section on Hopfield networks, no learning is performed by the network. Instead, a static set of weights define the stable states of the network. Finding the correct set of weights is a difficult problem, and how this was accomplished was not mentioned in this paper. In this case, also, the sensor was modeled using the elastic half-space assumption and a cylindrical indentor.

Chapter 3

Experimental Setup

As explained in the previous chapters, analytical models for tactile sensors are difficult to obtain. Numerical methods give a simple solution, but are difficult to invert. Neural networks can be trained to provide an accurate approximation of almost any function, and are ideal for inverting numerical data.

This chapter describes both the forward sensor model (FEA) and the inverse model (neural network), and explains the experiments conducted to examine the feasibility of using this approach to soive tactile sensing problems.

3.1 The Model

In the finite element analysis model, the elastic layer and indentor are modeled as a collection of finite elements sometimes called blocks. The corners of the blocks are called nodes. Within a single object, in this case the layer or the indentor, adjacent blocks share the nodes at common corners and are therefore attached together. The nodes are free to move within the model, constrained only by the assigned properties of the material being modeled and any additional constraints within the system (see Figure 13).

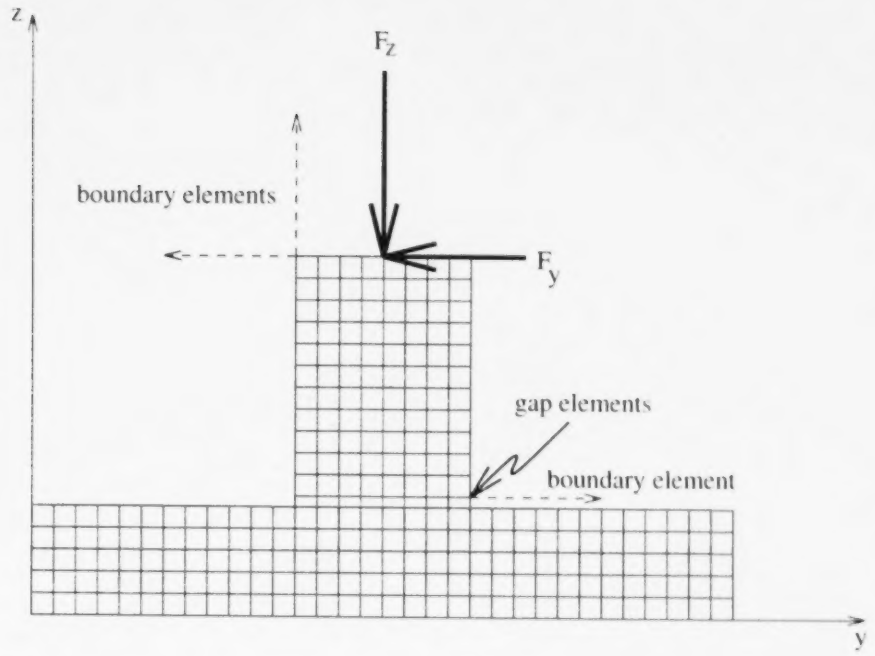


Figure 13: Diagram of the FEA model used to create training and testing data for the neural networks.

In the two-dimensional model[SZT93] used for experimentation in this thesis, three assumptions were made concerning properties of the materials used in the model. The first assumption is that of *isotropic media*. Each medium (the indentor and the elastic layer) is assumed to be homogenous and to have the same elastic properties independant of its orientation (or the orientation of the axes). This assumption is not strictly correct for a physical sensor, as the sensing elements embedded in the elastic layer are not of the same material as the layer itself. The second assumption is that of *linearity* where it is assumed that the second and higher order (non-linear) terms in strain calculation are insignificant [BMP65]. The validity of that assumption relies on the third assumptiuon, that of *small displacements*. If the displacements caused by the applied forces are small, the non-linear terms become insignificant. These assumptions impose a limit to the possible applied forces, as larger forces may cause a significant non-linear effect. Therefore the data generated by this model can be assumed valid for cases where these assumptions hold.

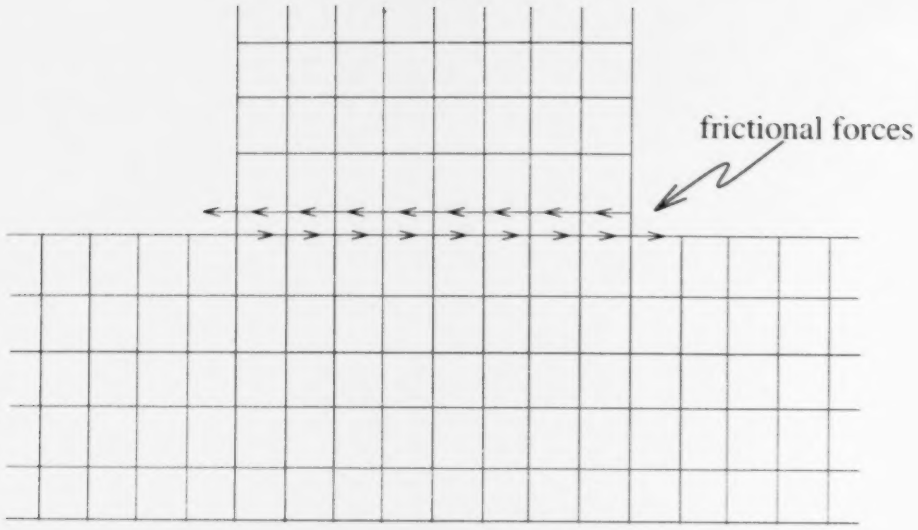


Figure 14: Interface area between indentor and elastic layer in the FEA model.

To produce the data used in training the networks, the elastic layer was modeled as a one-inch wide rubber strip, with certain values for the properties of elasticity and compressibility (see Appendix). The bottom of the layer was constrained to be held rigidly in place with respect to both axes by boundary elements. The indentor was modeled as steel with a different set of values for the material properties. Elastic boundary elements were placed at two opposing corners of the indentor. These were used to keep the indentor from gross movement with respect to the elastic layer, but acted as springs, allowing some smaller movement required for the functioning of the model.

In the interface between the two objects, gap elements [ALG90] were used to model object interaction and friction. Each gap element connected a single node of the elastic layer surface to a node of the indentor surface. While the z-axis distance between the nodes remained positive, the gap elements applied no forces to their connected nodes. If the distance became negative, the gap elements applied opposite forces to the two nodes in order to bring the distance to zero. Each pair of nodes also has a pair of opposing forces applied to them along the y-axis (see Figure 14). These forces were proportional to the normal force at that point and provided a simple

model for friction. The total frictional force was made to equal the applied tangential force as the system was in a state of static equilibrium.

Two forces were applied to the center of the top of the indentor. The system was then iterated until a sufficiently stable state was reached, and the stresses were calculated for the blocks in the bottom of the elastic layer.

Contact parameters considered for the experiments included the indentor size and position only. Gap elements were added or deleted to maintain the interface between the indentor and the elastic layer. The frictional forces were adjusted to reflect the different normal and tangential forces applied to the indentor.

3.2 The Network

A feed–forward neural network using the backpropagation learning algorithm was used as the inversion method for the experiments conducted in this thesis. Figure 15 shows the topology of the network and the connections between the nodes. The input layer has 16 nodes (labeled x_1 through x_{16}) which take the presented stress profile and propagates it to all of the hidden layer nodes. These nodes (numbered 17 through 32) process the inputs according to their internal weight vectors and pass their results on to the output layer. Nodes 33, 34 and 35 (labeled y_1 through y_3) accept the values calculated in the hidden layer and produce the three outputs of the network. The outputs of the network are the estimated normal force, the estimated tangential force, and possibly one of the width or position of the indentor. Although the FEA model produces 64 force readings (one for each block in the bottom level of the elastic layer), the network took only 16 inputs. The model outputs were grouped into adjacent fours and averaged together to emulate a sparser sensing element distribution. Using only 16 inputs greatly reduces the overall size of the network and allows for faster training times.

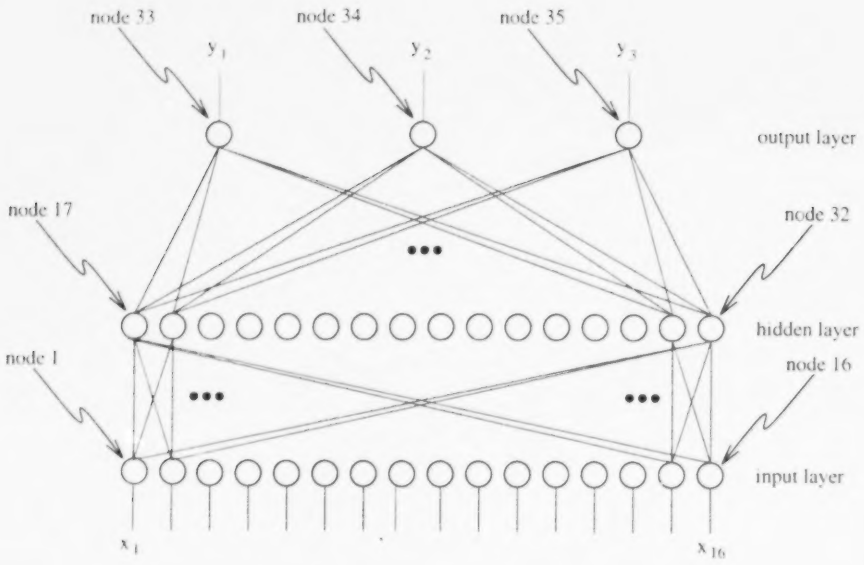


Figure 15: Layout of the backpropagation network.

The second (hidden) layer of the network consisted of 16 nodes, each receiving all of the inputs and processing them to produce an output value. The three output nodes take these values and produce the required outputs of the network.

A slightly modified backpropagation learning rule was used in training the network. A momentum term was added, causing the learning algorithm to take the previous adjustment to the weight vectors as the best direction for the current change, and use the current output errors to modify that direction for better error reduction. This algorithm tends to decrease the training times of backpropagation learning. The weight update equation (Equation 2.21) is changed to include the momentum term as follows:

$$\Delta w_{li_j}(t+1) = \alpha \delta_{li} O_{(l-1)_j} + \mu \Delta w_{li_j}(t) \tag{3.1}$$

where $\Delta w_{li_j}(t+1)$ is the change for weight w_{li_j} for update t and $\Delta w_{li_j}(t)$ is the previous update's weight change.

The backpropagation neural network used in the experiments was implemented

Parameter	*Value*	*Description*
α	0.2	learning rate
μ	0.8	momentum term
c	0.1	flat spot elimination constant

Table 3.2: Parameters for the backpropagation learning algorithm.

using the *Stuttgart Neural Network Simulator*. SNNS is a general purpose neural network simulator capable of simulating many different kinds of network with various learning algorithms. The standard BackPropMomentum learning algorithm was used, with the parameters shown in Table 3.2. These parameters were decided upon by experimentation, using the values that produced the fastest training time and minimum error for a given network and training data set. The *learning rate* specifies the step size used to move across the error surface. The momentum term dictates the amount of influence the previous weight change will have on the current weight change. The flat spot elimination constant, c, changes the δ calculation (Equation 2.20) to

$$\delta_{li} = s'(I_{li} + c) \sum_{k=1}^{M_{(l+1)}} w_{(l+1)ki}\delta_{(l+1)k} \tag{3.2}$$

allowing the learning algorithm to avoid flat areas on the error surface. The error surface can be thought of as an M dimensional surface where M is the number of weights in the network. The height of the surface for any given combination of weights corresponds to the average network error. For a more detailed explanation of these parameters, see [Gro93].

As the network only accepts inputs in the range of zero to one, the outputs of the FEA model were scaled by a constant factor to map them into this range. For each experiment, a set of data was created, with half being used for network training and half for network testing. The network was presented the training data in a random order and the learning algorithm was applied. Then the testing dataset was presented and the sum of the quadratic differences between the correct and estimated outputs

for all the output nodes was calculated. This process was repeated until the error dropped below a pre-determined value.

3.2.1 Scaling

One significant problem with using backpropagation neural networks is that of scalability. As the problem size increases (perhaps by requiring the recovery of another parameter) the number of nodes in the network will generally also increase. This will cause both an increase in the time needed to train the network and the time needed for the network to calculate its approximation of the required functions. The time required for a forward pass through a three-layer feed-forward network can be expressed as a function of the number of input (i), hidden (h) and output (o) nodes. For each connection into a hidden or output layer node, one addition/multiply pair must be executed. Then, each hidden and output node must compute the sigmoid function $s(I) = 1/(1 + e^{-I})$. The following equations show the number of operations:

$$additions = ih + ho + h + o \tag{3.3}$$

$$multiplications/divisions = ih + ho + h + o \tag{3.4}$$

$$exponentials = h + o \tag{3.5}$$

For the network used in experiments 2 and 3, these values are 323, 323 and 19 respectively. Unfortunately, while the increase in the network computation time is linear, the training time will increase exponentially with the addition of new outputs. A new variable will increase the dimensionality of the input space and if a representative sample of the input space is to used for training, the training time will increase exponentially. Cautions must be taken when applying the proposed approach to a general problem unless measures are taken to parallelize the computations.

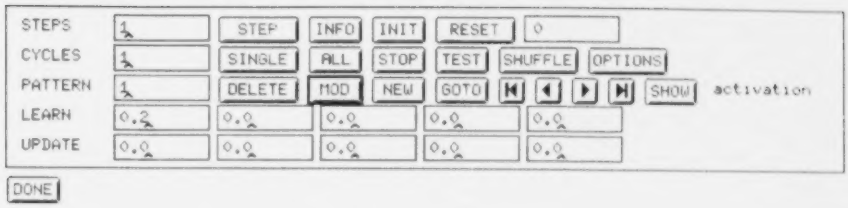


Figure 16: SNNS "remote" panel. This panel allows the training and testing of networks, and the adjustment of the various training parameters.

3.2.2 The Simulator

The *Stuttgart Neural Network Simulator*[Gro93] was developed at Institute for Parallel and Distributed High Performance Systems in Stuttgart, Germany. The goal of the project was to create an efficient and flexible simulation environment for research on and application of neural nets. This simulator was used in all the experiments conducted in this thesis.

SNNS includes a graphical user interface that runs under the X-window system. This user interface can be used to create, train and test networks, and allows a graphical exploration of the network, its weights and internal values. Network building and training may also be done outside of the GUI, allowing networks to be trained off line. Tools also exist for converting the network to a C-language module that can be easily added to a program.

The "remote control panel" (Figure 16) controls the parameters used in network training, and allows training and testing to be performed on a network. Training and testing patterns can be presented to the network singly or in groups and can optionally be followed by network learning. The order that patterns are presented to the network can be randomized which reduces selective learning that sometimes occurs in networks.

Figure 17 shows the activation values for each of the processing nodes in a network after an input pattern has been presented. A value is represented both as a number

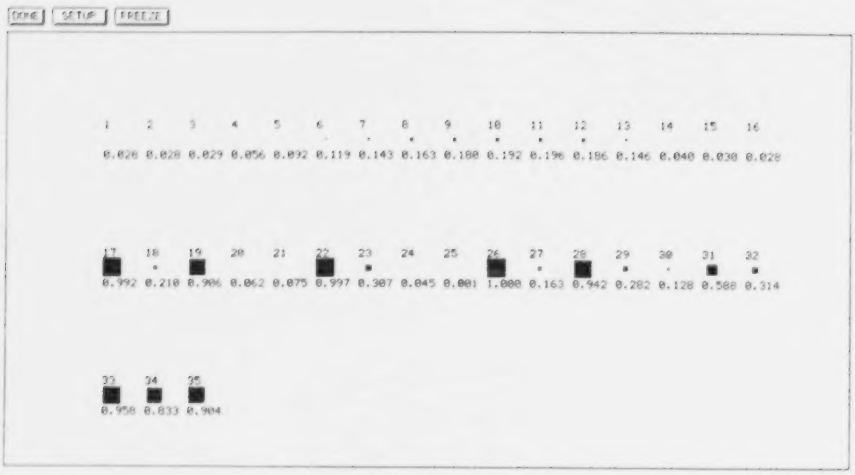


Figure 17: SNNS network activation values display. The output of each processing node in the network is represented in both a numeric and graphical form.

and as a graphical square. The square changes size according to the magnitude of the node's output. The first row of values (labeled 1 through 16) are the input nodes of the network, and the values represent an input pattern (stress profile). The second row of nodes are the hidden nodes (17 through 32) whose output values are shown. The last three nodes correspond to the output values of the network, in this case the two contact forces and indentor width (scaled to the range $[0, 1]$).

The weight matrix of a network defines the function being performed by the network. Figure 18 is a representation of the weights of a trained feed-forward network. This representation is called a Hinton diagram [Gro93]. The network is shown as a matrix of squares with the size of a square representing the magnitude of the corresponding weight and the color of the square representing the sign (black for a positive weight and white for a negative weight). The square at the intersection of row i and column j represents the weight w_{ij} of the neural network, where w_{ij} is the weight from node i to node j. By interpreting the patterns in the weights, some understanding of how the network functions can be obtained.

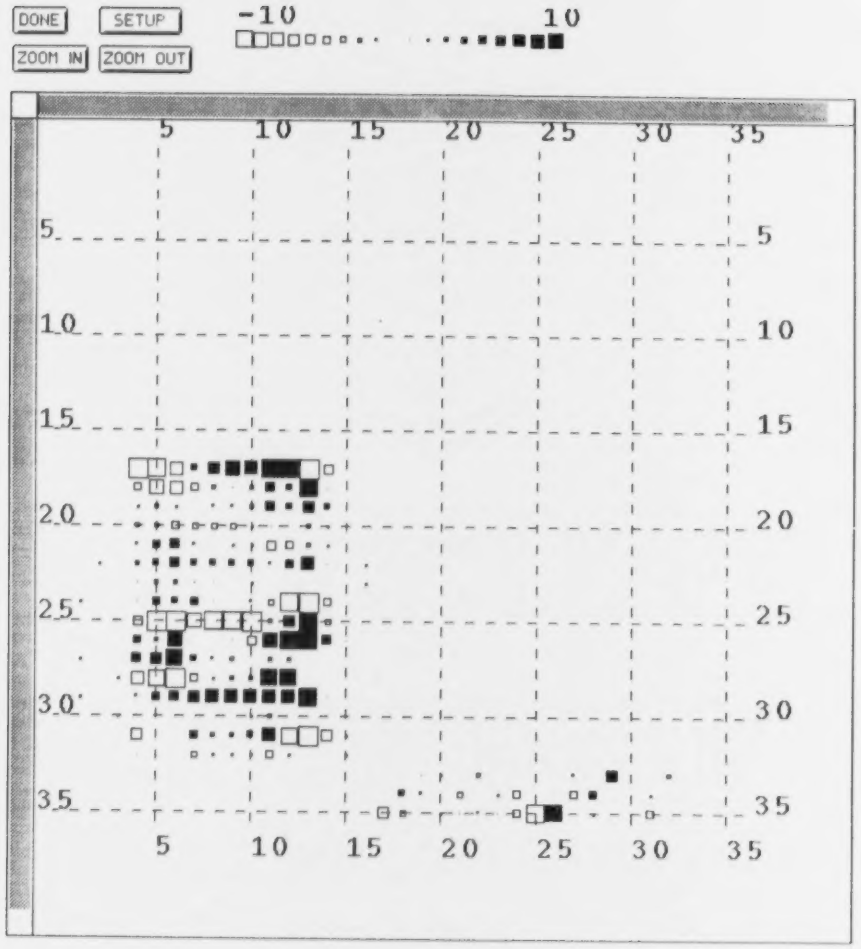


Figure 18: SNNS Hinton diagram for displaying network weight values.

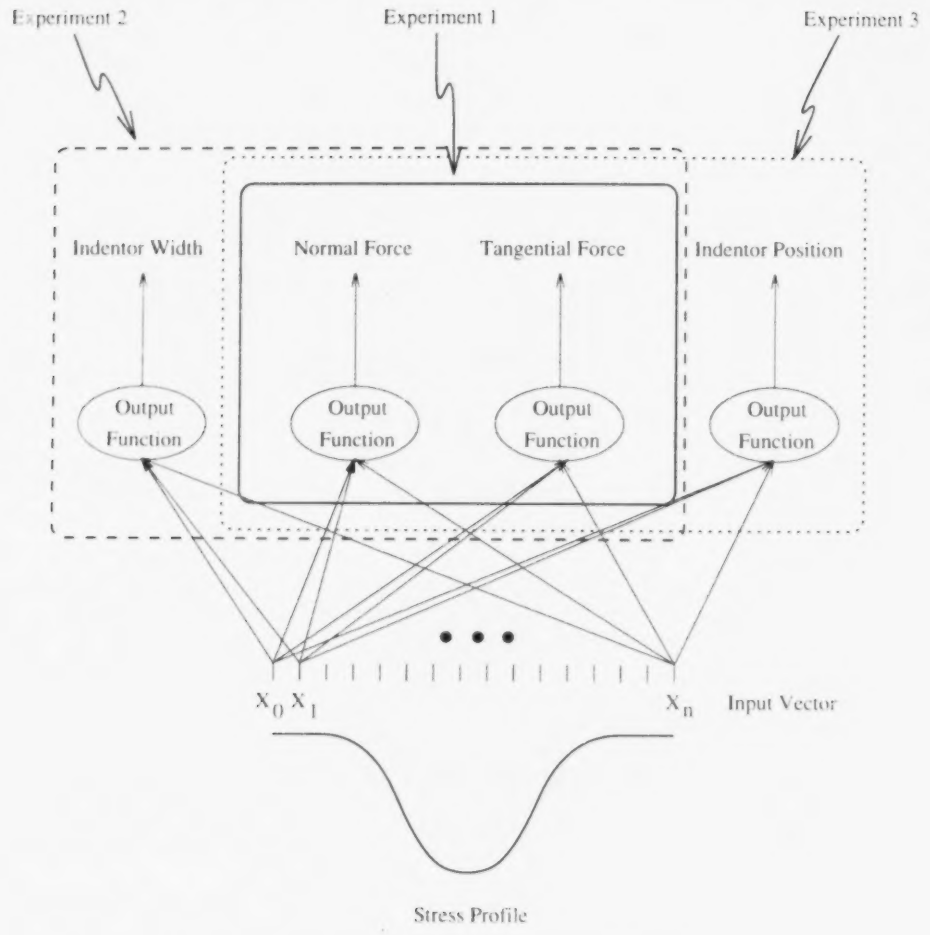


Figure 19: Overview of the experiments.

3.3 The Experiments

A preliminary experiment, aimed at judging the feasibility of using neural networks for interpreting tactile data, was conducted using profiles generated by the FEA model. These profiles corresponded to contact situations where only normal force was applied to the indentor. The tangential force was set to zero, and the indentor width and positions were held constant. While the results of this experiment are not shown, the performance of the neural network was confirmed, and three more extensive experiments were conducted (Figure 19). The contact parameters that were varied in each

	Parameters and Parameters Ranges							
	Width	Δ*Width*	*Pos.*	Δ*Pos.*	F_t	ΔF_t	F_n	ΔF_n
Exp 1	7–14 mm	0.8 mm	0 mm	n/a	0.2–2.2 N	0.2 N	4.4–24.4 N	2.2 N
Exp 2	7–14 mm	0.8 mm	0 mm	n/a	0.2–2.2 N	0.2 N	4.4–24.4 N	2.2 N
Exp 3	11 mm	n/a	−4.0–4.0 mm	0.8 mm	0.4–1.4 N	0.1 N	4.4–13.3 N	4.4 N

Table 3.3: Summary of contact parameters used in the three experiments

of the three experiments are shown in Table 3.3. In these experiments, the variables that changed were calculated by the network, except for the first experiment where only the contact forces were calculated.

In first two experiments, the normal force was again varied from 4.4 to 24.4 Newtons in increments of 2.2 Newtons. The tangential force was varied from 0.2 to 2.2 Newtons in increments of 0.2 Newtons. The width of the indentor was also varied, in order to determine if the network would generalize over different indentor widths. Indentor position was held static for both experiments. In experiment one, two output nodes of the network were used to recover the applied forces, while the indentor width was not calculated. This experiment was meant to show the generalization properties of the network, as it should generalize the recovery of the contact forces over several indentor widths. Figure 20 shows the stress profiles for several tangential forces and two indentor widths at a single normal force. The second experiment attempted the recovery of the indentor width as well as the contact forces.

The last experiment used a smaller set of normal and tangential forces, and only a single indentor width. The indentor was then moved across the surface of the elastic layer, with data being generated for a number of different positions. The network was required to recover the contact forces and the indentor position.

The outputs of the trained networks were collected and compared to the actual values of the variables being calculated. The quadratic differences between the actual and calculated values were plotted to show the error levels of the networks for different

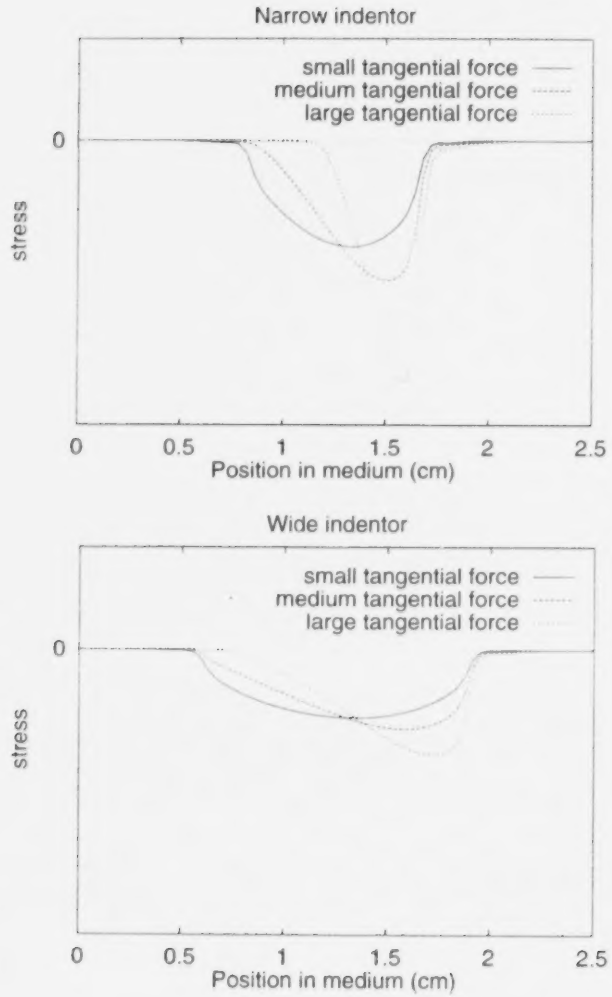


Figure 20: Stress profiles for different indentor widths and tangential forces.

contact parameters. These results are shown in Chapter 4.

3.3.1 Physical Experiment

To show the applicability of this approach to physical sensors, an *Interlink* Force Sensing Resistor was employed to measure force profiles for various applied normal forces, ranging from one to nineteen Newtons in steps of approximately one Newton. The sensor consisted of an array of 16x16 tactile elements. This array was covered by a thin rubber layer and was impinged upon by a steel block . The force was applied to the block using a robotic arm, with a force sensor attached to the wrist to measure the actual forces. The obtained stress profiles were used to train the same network used in the previous experiments and the network was tested against more data produced by the physical sensor.

Chapter 4

Simulation and Experimental Results

This chapter provides the results of both the experiments done on the FEA model produced data, and the experiment done on data for the physical sensor. The input stress profiles generated by the FEA model provide a variety of different sets of contact situations, with variations in several variables within the same profile. For instance, Figure 21 shows a set of stress profiles where the applied normal and tangential forces are increasing together so that the applied tangential force is always one tenth of the applied normal force.

4.1 Simulation Results

4.1.1 Normal and Tangential Force Recovery

In this experiment, the applied normal and tangential contact forces were recovered from stress profiles where the contact forces and the indentor width were varied across ten values each. Half of the profiles (500 profiles) were used to train the network,

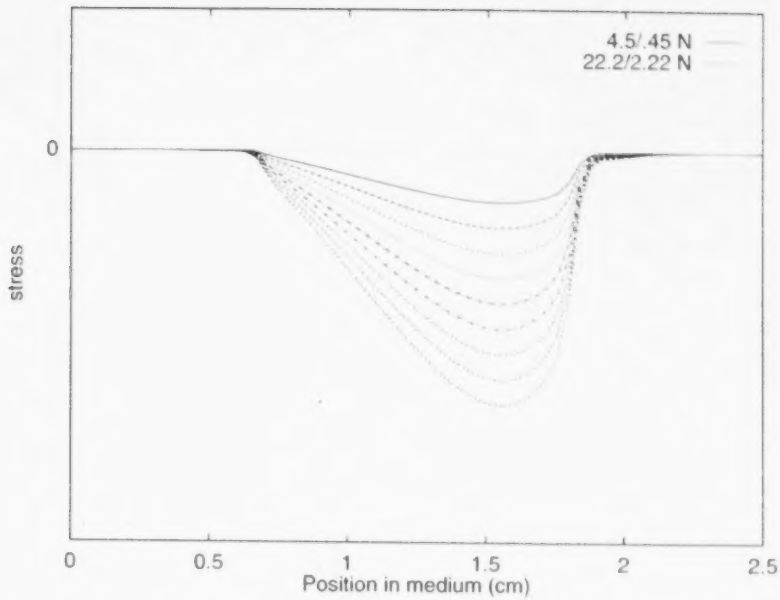


Figure 21: Stress profile for varying applied normal and tangential forces.

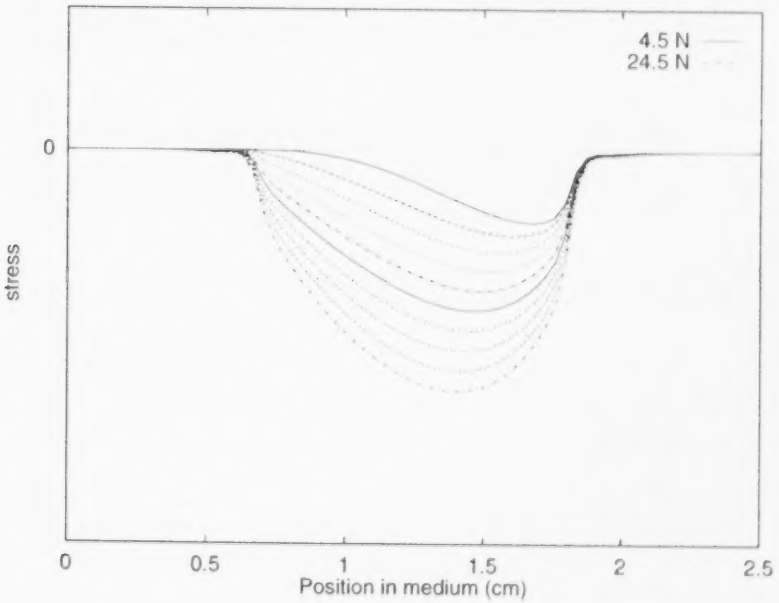


Figure 22: Stress profiles for varying normal force with other variables held static.

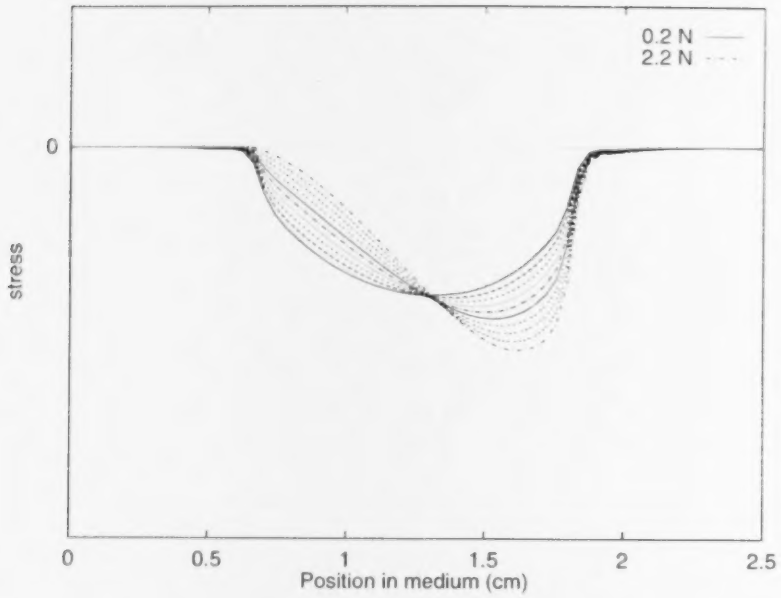


Figure 23: Stress profiles for varying tangential force with other variables held static.

and the other half to test the performance of the network after training. Samples of the stress profiles for different normal and tangential contact forces can be found in Figures 22 and 23.

Network Performance in the first experiment

After training the network for 5000 iterations through the training set, where a typical iteration time is 0.4 seconds, the average error (quadratic difference of the expected versus computed outputs) of the network outputs had dropped below 10%. Figures 24 through 26 depict the relative errors in the two outputs of the network. The vertical axis of the graphs shows the relative error while the base axes show the two other variables. The individual error values are all below 15%, and most are below 5%. The worst errors occur at low normal forces coupled with high tangential forces and narrow indentor widths (Figures 24 and 26). At these points in the input space, the stress profiles become very narrow and unlike the profiles at other normal and

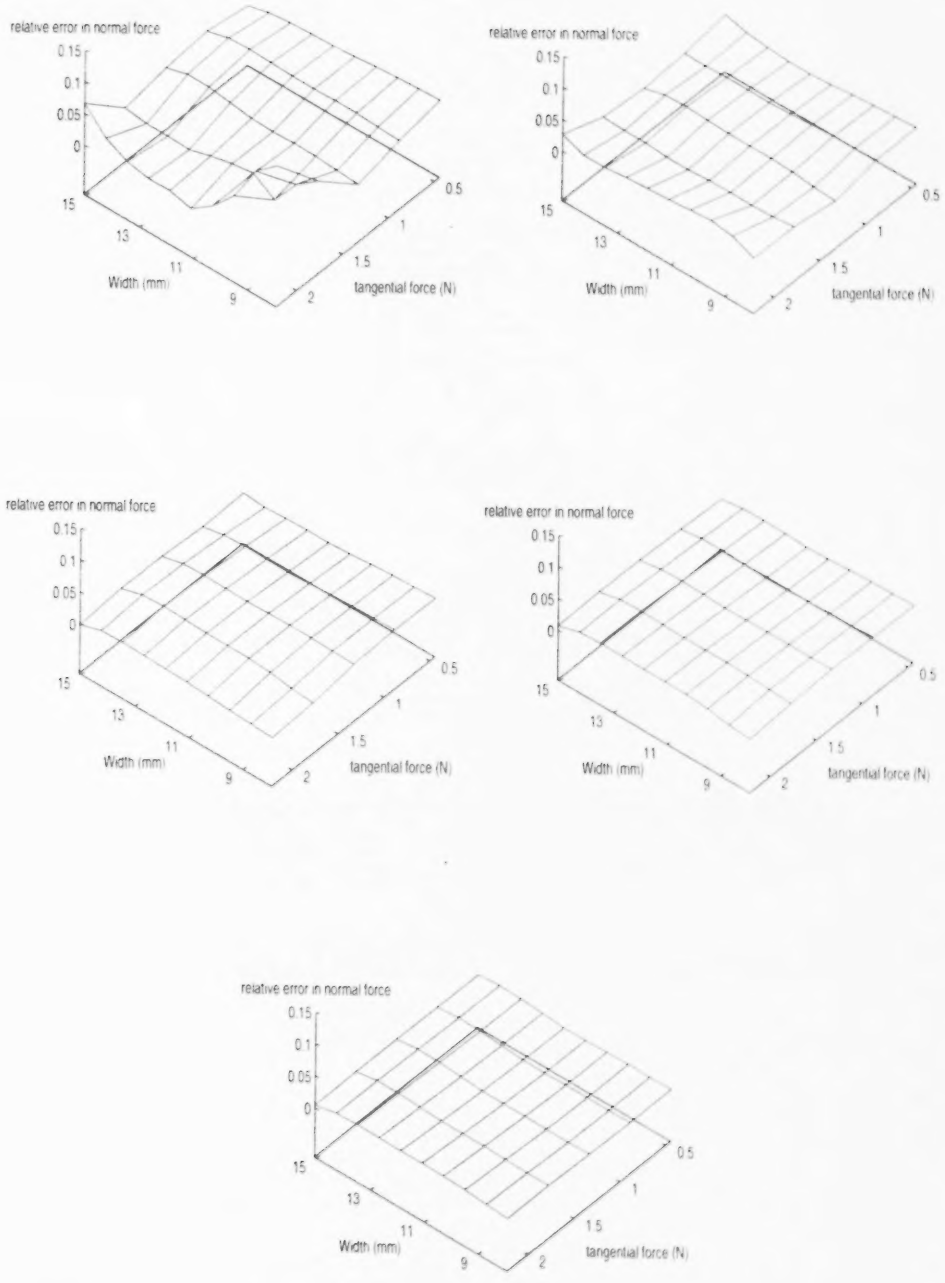


Figure 24: Relative error in normal force. Each graph represents the error in normal force recovery for a given applied normal force. The force values for this figure range evenly from 4.5 N in the top left corner to 13.3 N at the bottom.

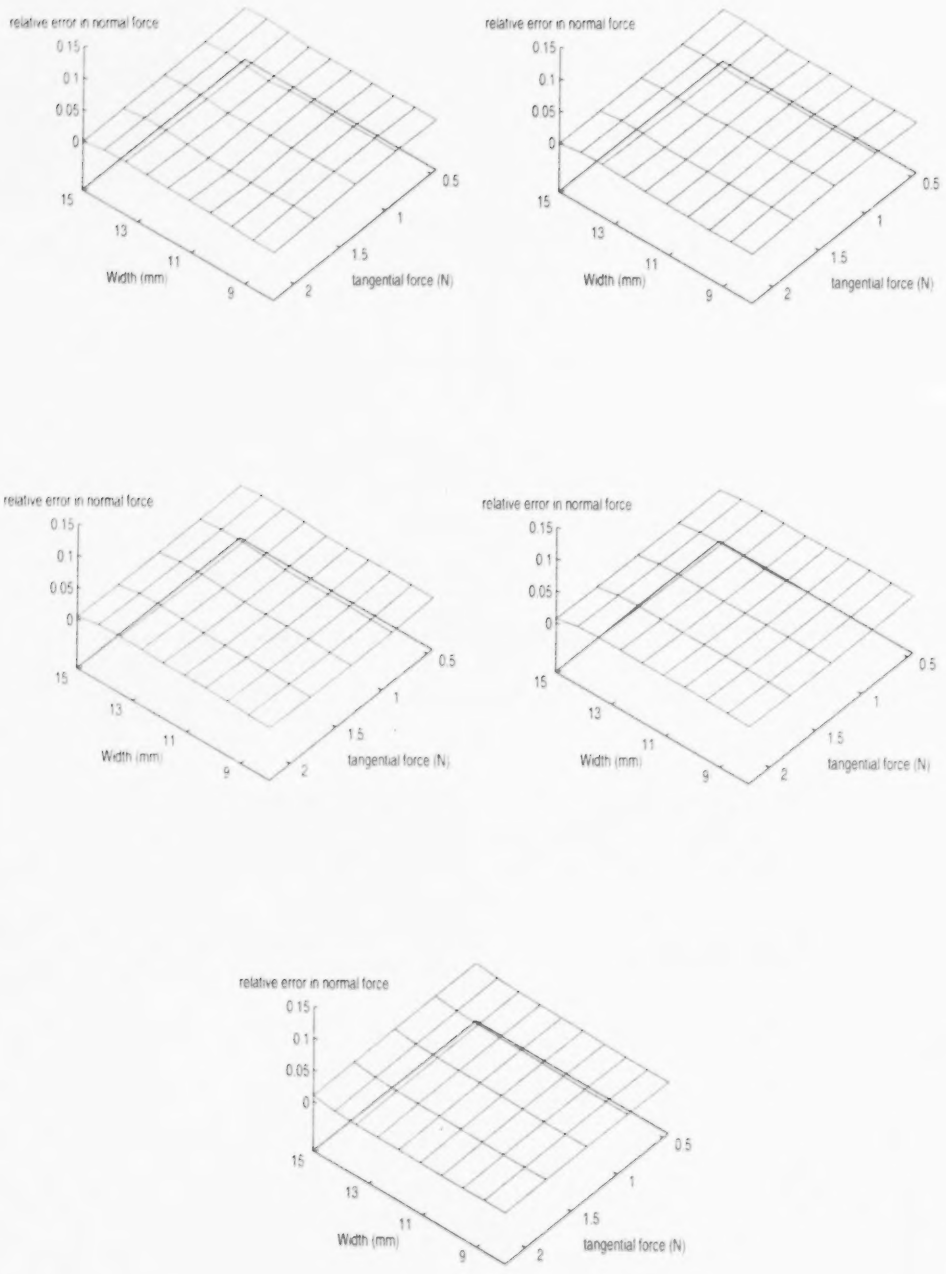


Figure 25: Relative error in normal force. Each graph represents the error in normal force recovery for a given applied normal force. The force values for this figure range evenly from 15.6 N in the top left corner to 24.5 N at the bottom.

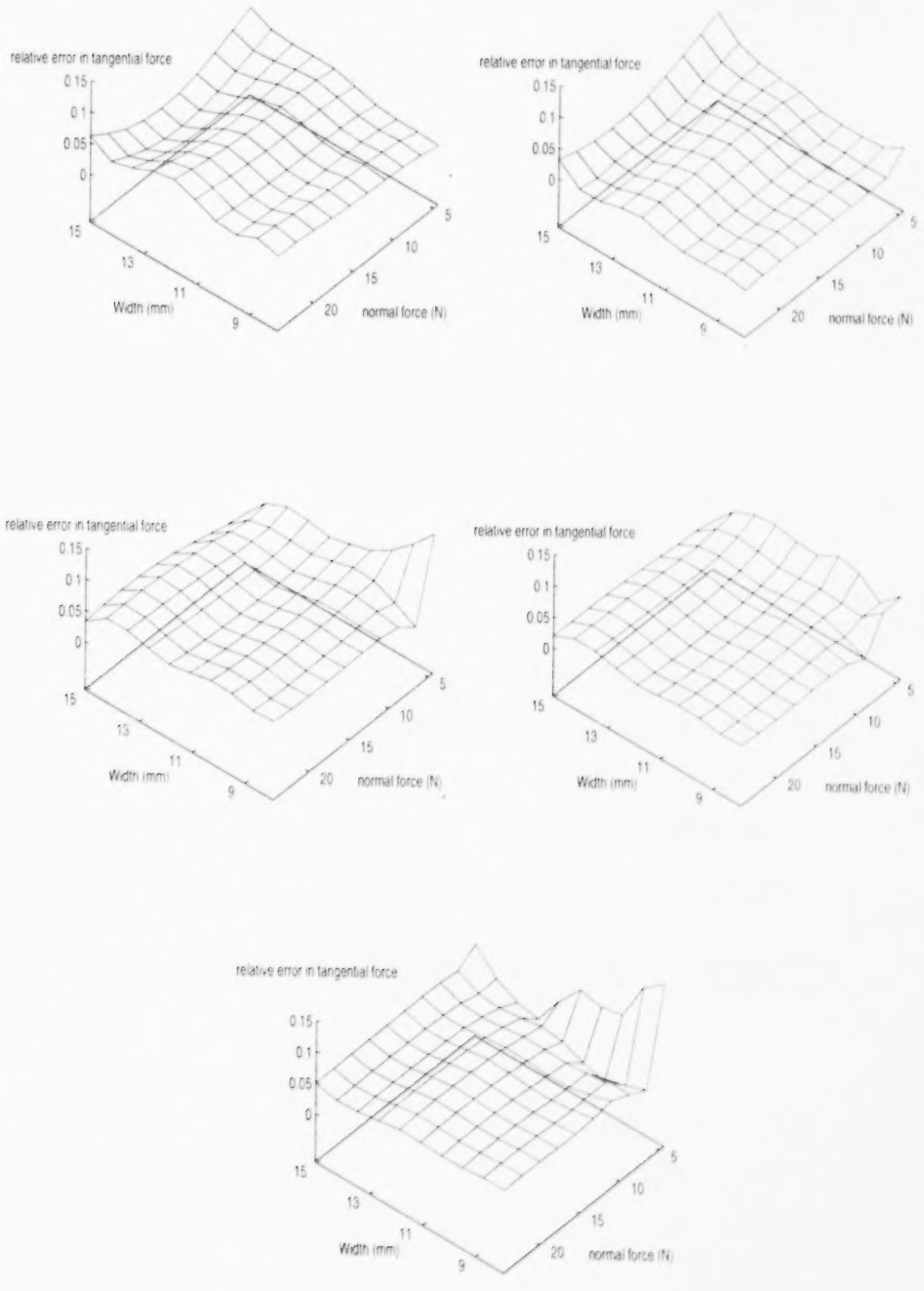


Figure 26: Relative error in tangential force. Each graph represents the error in tangential force recovery for a given applied normal force. The force values for this figure range evenly from 0.45 N in the top left corner to 2.2 N at the bottom.

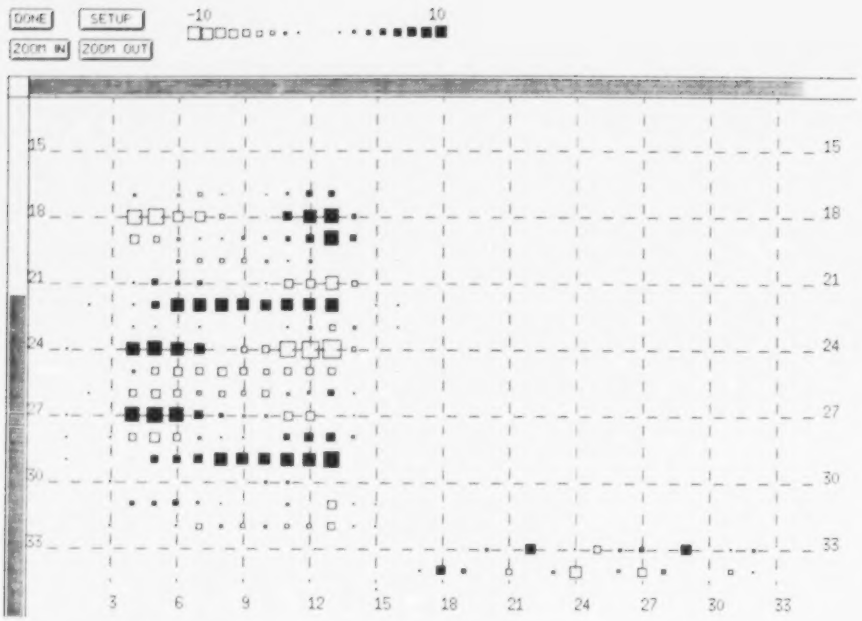


Figure 27: Hinton diagram of the trained weight values for the network recovering applied normal and tangential forces only.

tangential forces. It is suggested that an increase in the amount of training data provided in this range could alleviate this problem.

Weight Analysis for the first experiment

The Hinton diagram shows the weight values for the inputs to each processing node in the network. Figure 27 shows the weights for the trained network used in the first experiment. Processing nodes 17 through 32 are the hidden layer nodes, and nodes 33 and 34 are the output nodes. The two output nodes represent normal force output and tangential force output, respectively.

The normal force output has large positive weights on the outputs of nodes 22 and 29, and a negative weight on the output of node 25. Both of the positive weighted nodes have a large positive weight associated with the majority of the input nodes. The negatively weighted node applies a negative weight to the entire input vector,

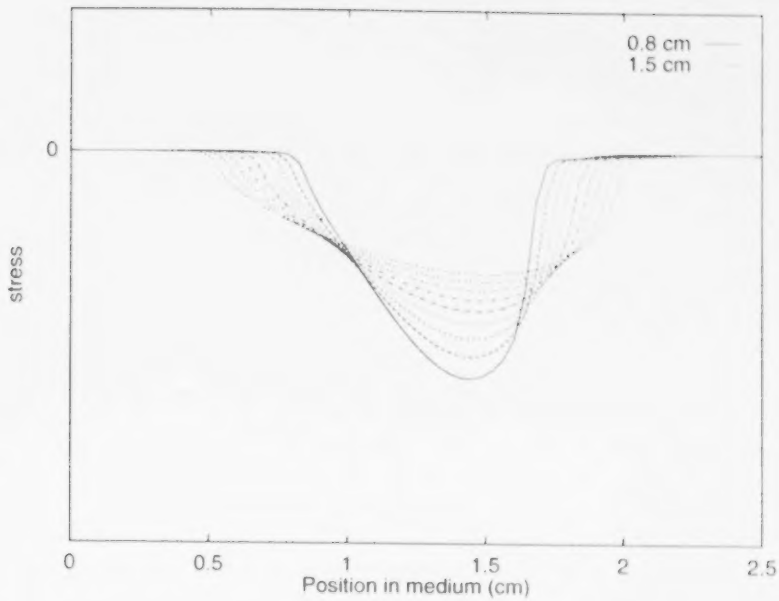


Figure 28: Stress profiles for varying indentor widths with the other variables held constant.

having the same effect as the other two. This roughly equates to an integration over the input vector to determine the applied normal force [CZR94].

The tangential force output has a positive weight on node 18 and negative weights on nodes 24 and 27. The latter two have a positive weight on the left side of the input vector and a negative weight on the right side, while node 18 is the opposite. This amounts to a measure of the skew of the profile about the center of the sensor.

As can be seen from these results and the example stress profiles, the network is recognizing and quantifying the changing features of the profiles.

4.1.2 Width Recovery

The stress profiles used in the training and testing of the network in this experiment were identical to those used in the previous experiment. In addition to the contact forces, the network was trained to recover the indentor width. Figure 28 shows the

change in the stress profiles with the varying of the indentor width, while the normal and tangential forces are held constant at 15.4 Newtons and 0.8 Newtons, respectively.

Network Performance for the second experiment

As in the previous experiment, the network was trained for 5000 iterations through the training data. The average error after training was slightly higher than in the first experiment (approximately 10%) but this was not unexpected with the addition of a third output function to learn. The relative error in the network outputs for each of the testing inputs is given in Figures 29 through 33. As in the previous experiment, a larger tangential to normal force ratio produces larger network error levels in the recovery of the applied normal and tangential forces. The recovery of width proved quite successful, with an average error near 1%. This function also experienced increased error levels for the extreme case of large tangential and small normal force as can be seen in Figure 32.

Weight Analysis for the second experiment

Figure 34 shows the weight values for the processing nodes in the network for the second experiment. The layout of the diagram is the same is in the previous example with an additional output node representing the indentor width (node 35). In this case, the normal force output relies heavily on a single hidden node (node 29) that performs a rough sum of the inputs as in the previous experiment. The tangential force output behaves in a manner similar to the previous experiment, relying on several hidden nodes that compute a skewing type function.

The network's derivation of the width output is a bit more complicated. The two hidden nodes (17 and 25) contributing large negative factors to the width output are computing similar functions of the inputs. They both have a large negative weight on the left side of the input vector and combined have a slightly positive weight to

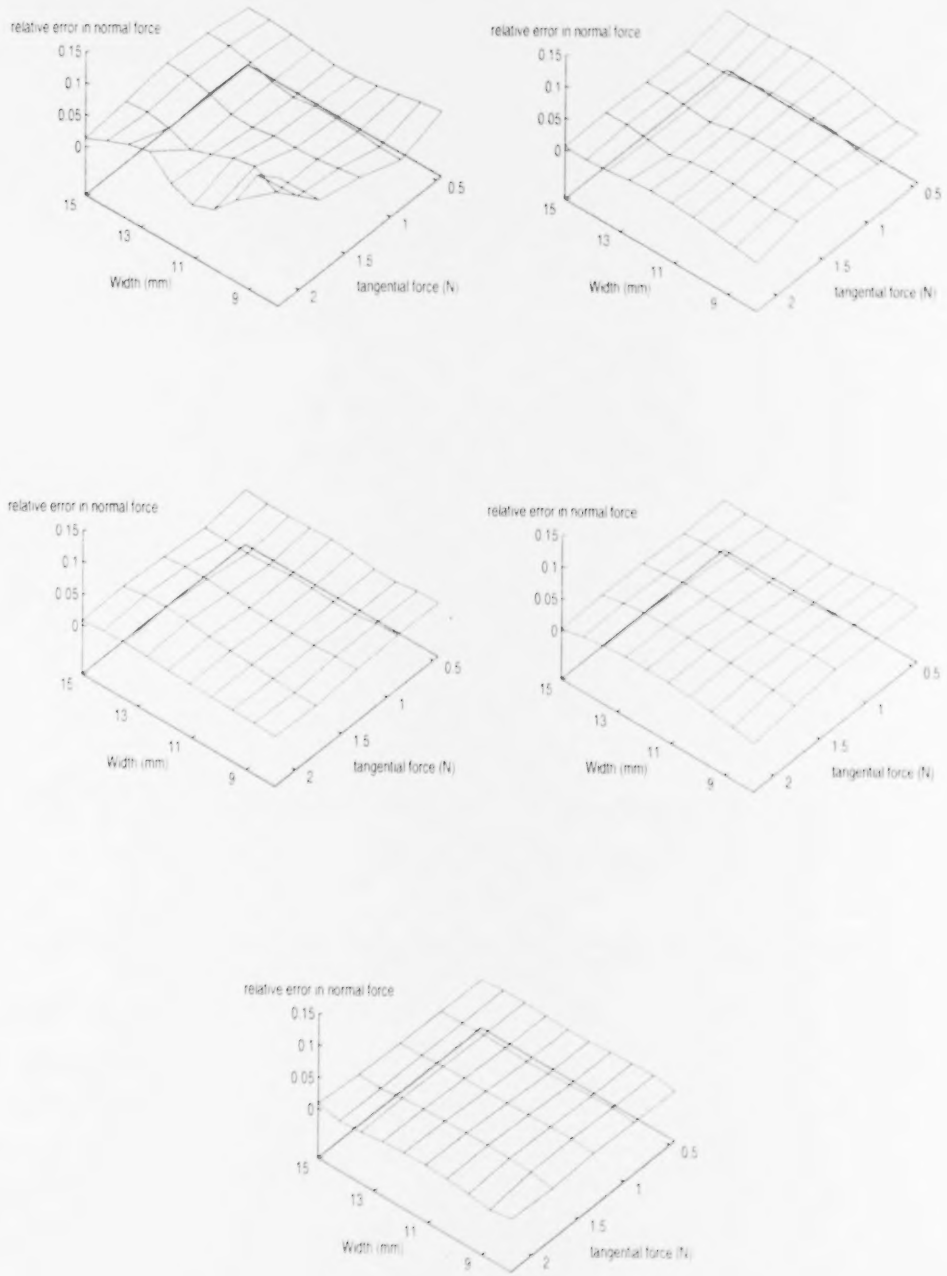


Figure 29: Relative error in normal force. Each graph represents the error in normal force recovery for a given applied normal force. The force values for this figure range evenly from 4.5 N in the top left corner to 13.3 N at the bottom.

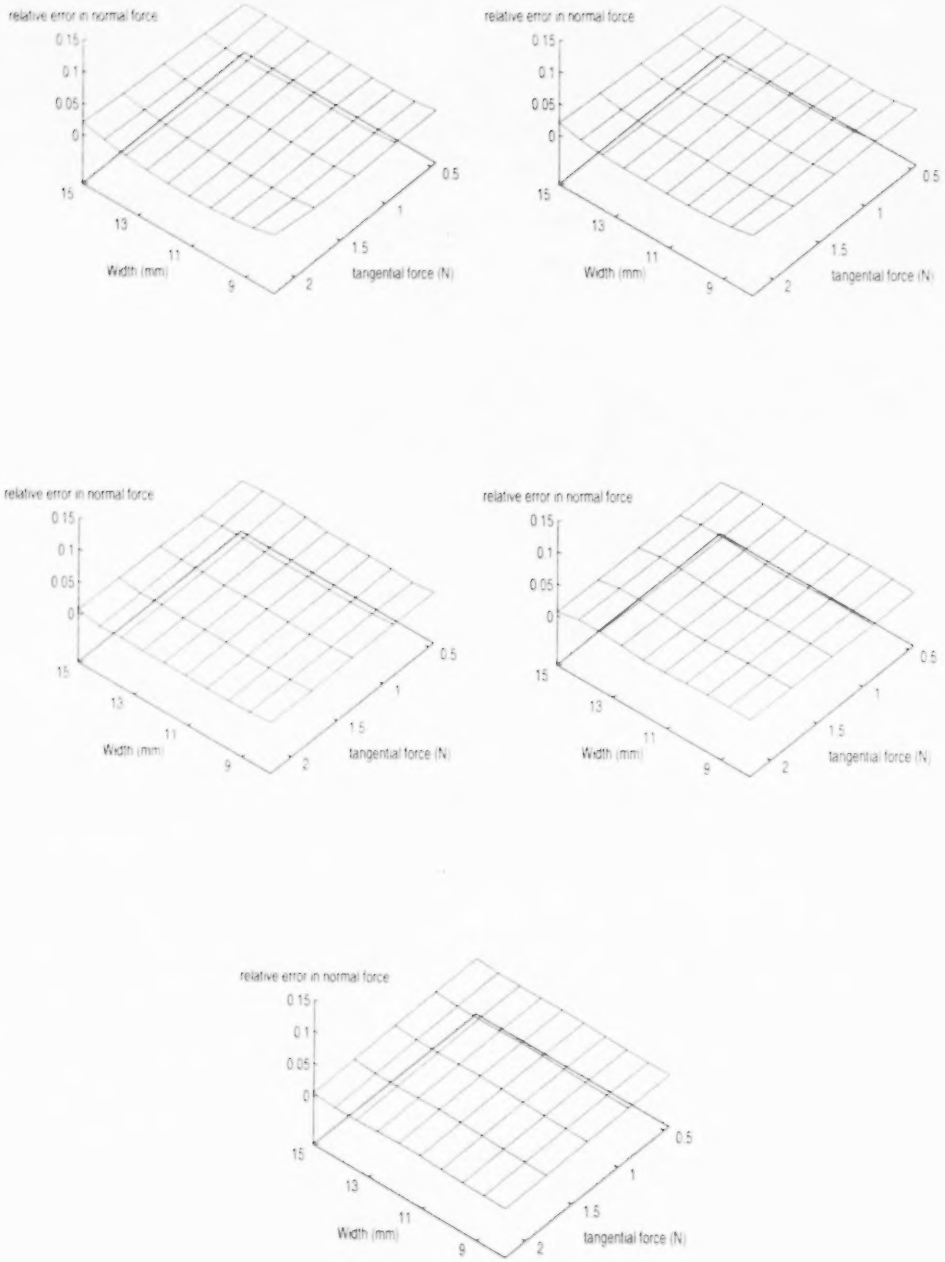


Figure 30: Relative error in normal force. Each graph represents the error in normal force recovery for a given applied normal force. The force values for this figure range evenly from 15.6 N in the top left corner to 24.5 N at the bottom.

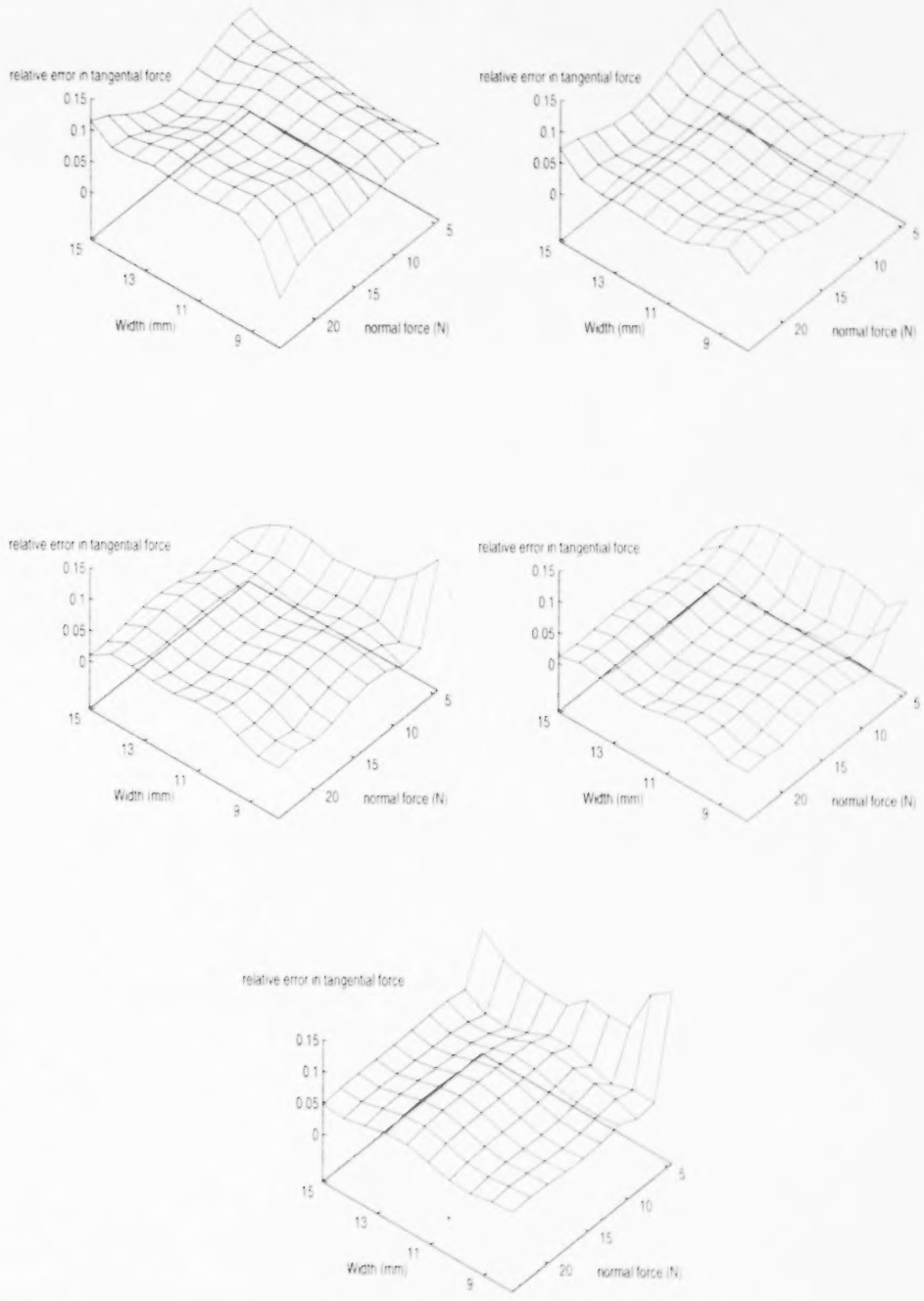


Figure 31: Relative error in tangential force. Each graph represents the error in tangential force recovery for a given applied normal force. The force values for this figure range evenly from 0.45 N in the top left corner to 2.2 N at the bottom.

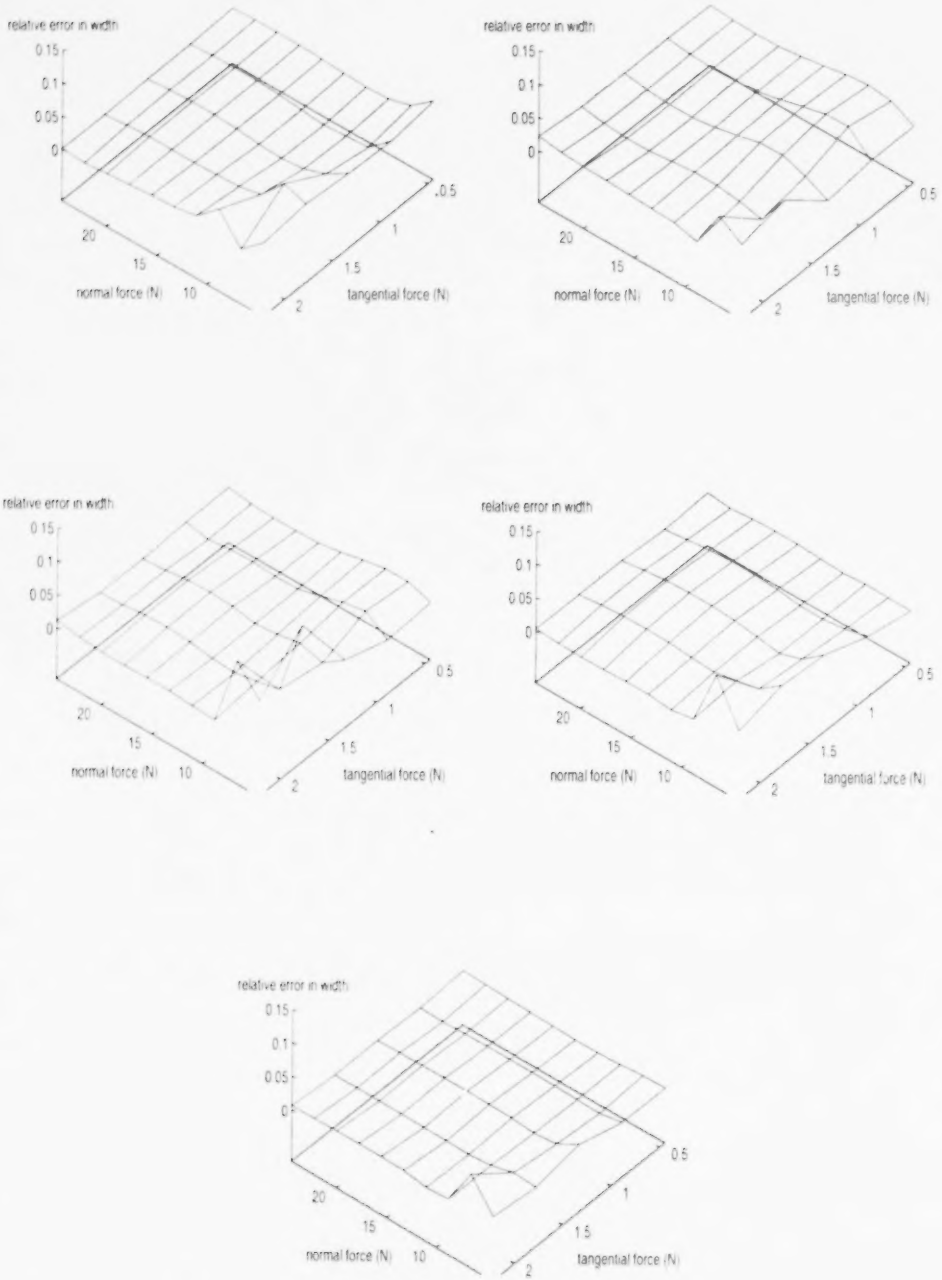


Figure 32: Relative error in indentor width. Each graph represents the error in indentor width recovery for a given width. The widths for this figure range evenly from 0.8 cm in the top left corner to 1.1 cm at the bottom.

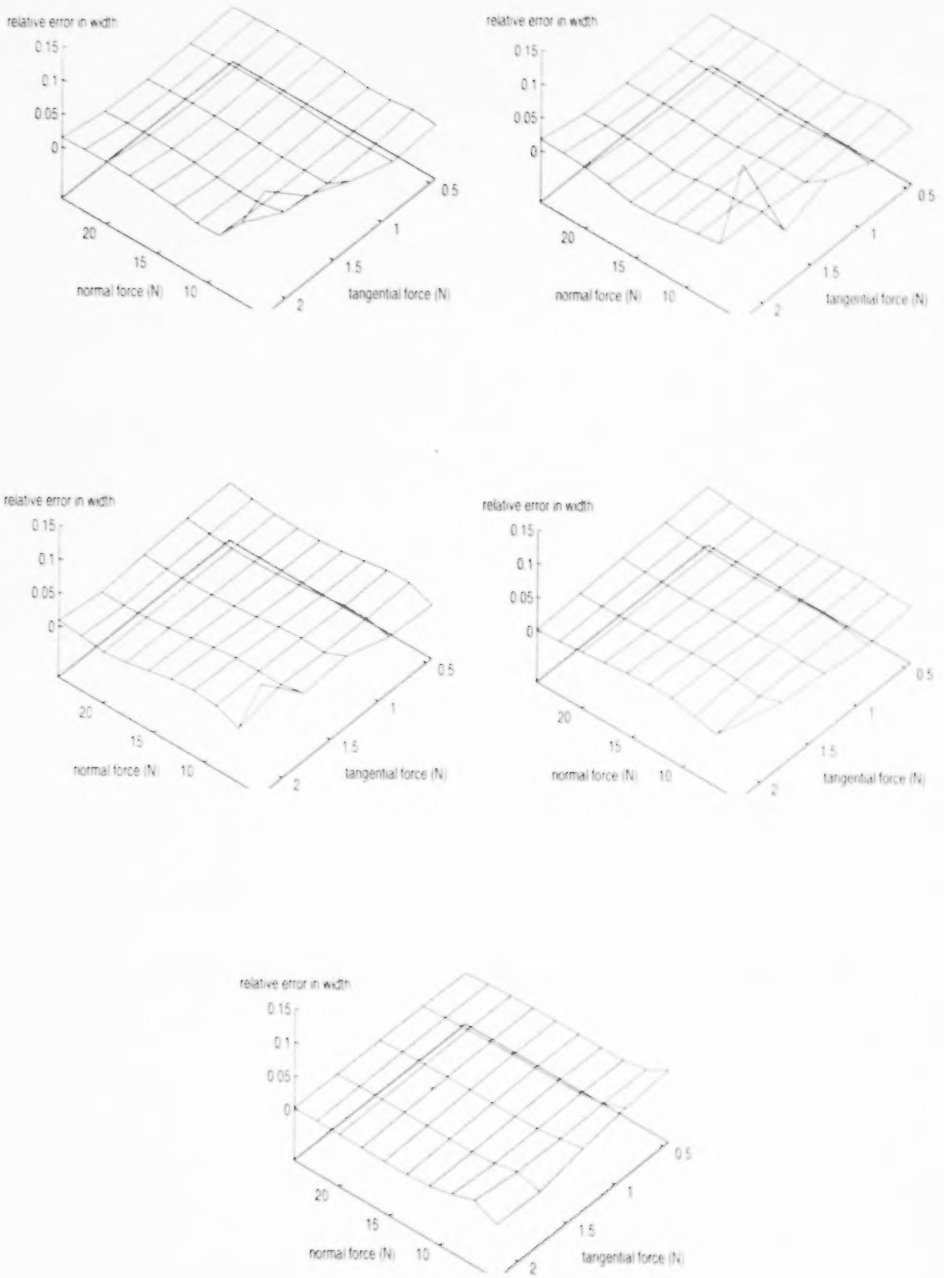


Figure 33: Relative error in indentor width. Each graph represents the error in indentor width recovery for a given width. The widths for this figure range evenly from 1.2 cm in the top left corner to 1.5 cm at the bottom.

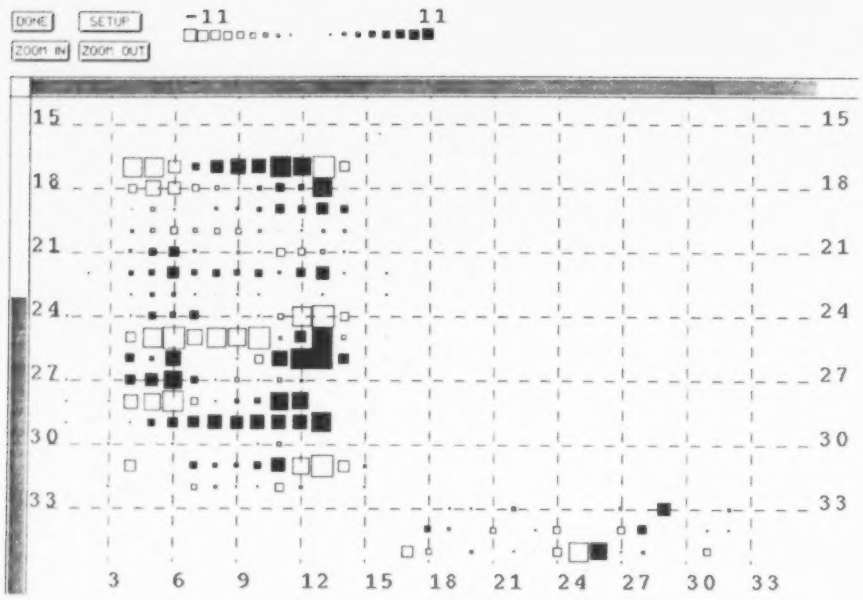


Figure 34: Hinton diagram of the trained weight values for the network recovering contact forces and indentor widths.

the rest of the vector. Node 26 has a large positive weight assigned to the left side of the input vector, a larger positive weight assigned to the right side, and no other significant weight. The positive effect of node 26 on the output makes the overall output function resemble a sort of bowl on the input vector with higher weights assigned toward the edges of the stress profile, and smaller weights assigned toward the center. This has the effect of measuring the width of the indentor.

4.1.3 Position Recovery

In this third experiment, a different set of stress profile inputs were used in the training and testing. Only three of the previous normal force values were used in the new data. The tangential forces ranged evenly between 0.45 N and 1.3 N. Figure 35 shows the effect of varying the indentor position on the stress distribution for a typical pair of applied normal and tangential forces. The positions were varied equally to both sides of center in increments of 0.8 mm, resulting in a range of -4.0 mm to 4.0 mm.

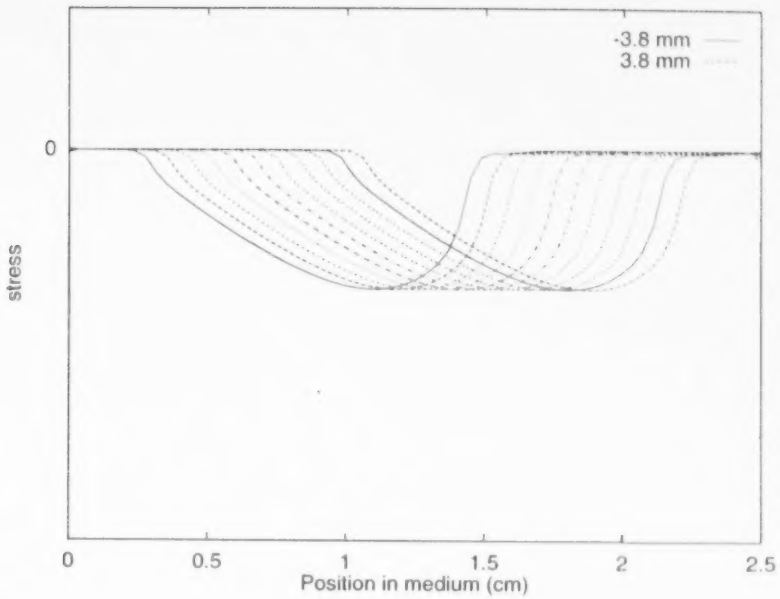


Figure 35: Stress profiles for varying indentor position with other variables held static.

Network Performance for the third experiment

The training time required to achieve low network errors in the case of the third experiment was considerably longer than in the previous two experiments (five hours as opposed to 50 minutes). After the short training time used in the other experiments, the average network error was acceptable (below 10%) but the majority of this error was due to incorrect tangential force estimation. After a longer training time (100000 iterations) the error in tangential force recovery was acceptable, and the results are shown in Figures 36, 38 and 39. The relative errors in normal force recovery and indentor position are below 1% in nearly all cases. The relative errors for tangential force (Figure 37) are below 5% in all cases except the largest tangential force which has errors as high as 9%. From the weight analyses in the previous sections, the tangential force recovery depended on a quantification of the skew of the stress profile about the center of the indentor. In this case, the center of the indentor is not in a static position, making this skew value much harder to compute.

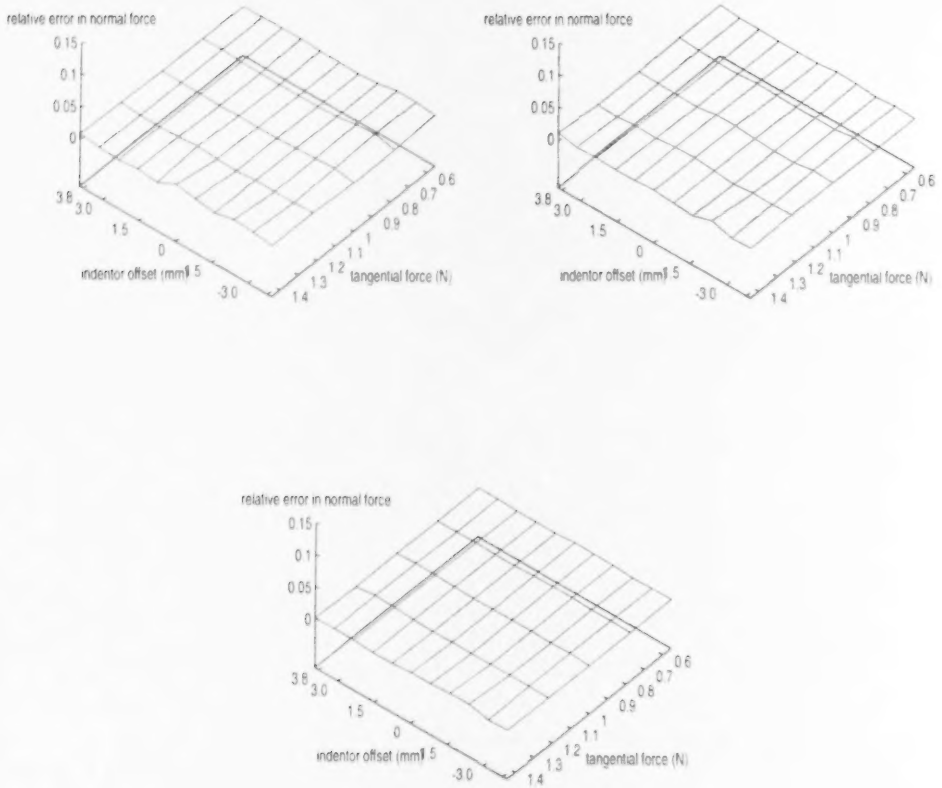


Figure 36: Relative error in normal force. Each graph represents the error in tangential force recovery for a given applied normal force. The force values for this figure range evenly from 0.45 N in the top left corner to 2.2 N at the bottom.

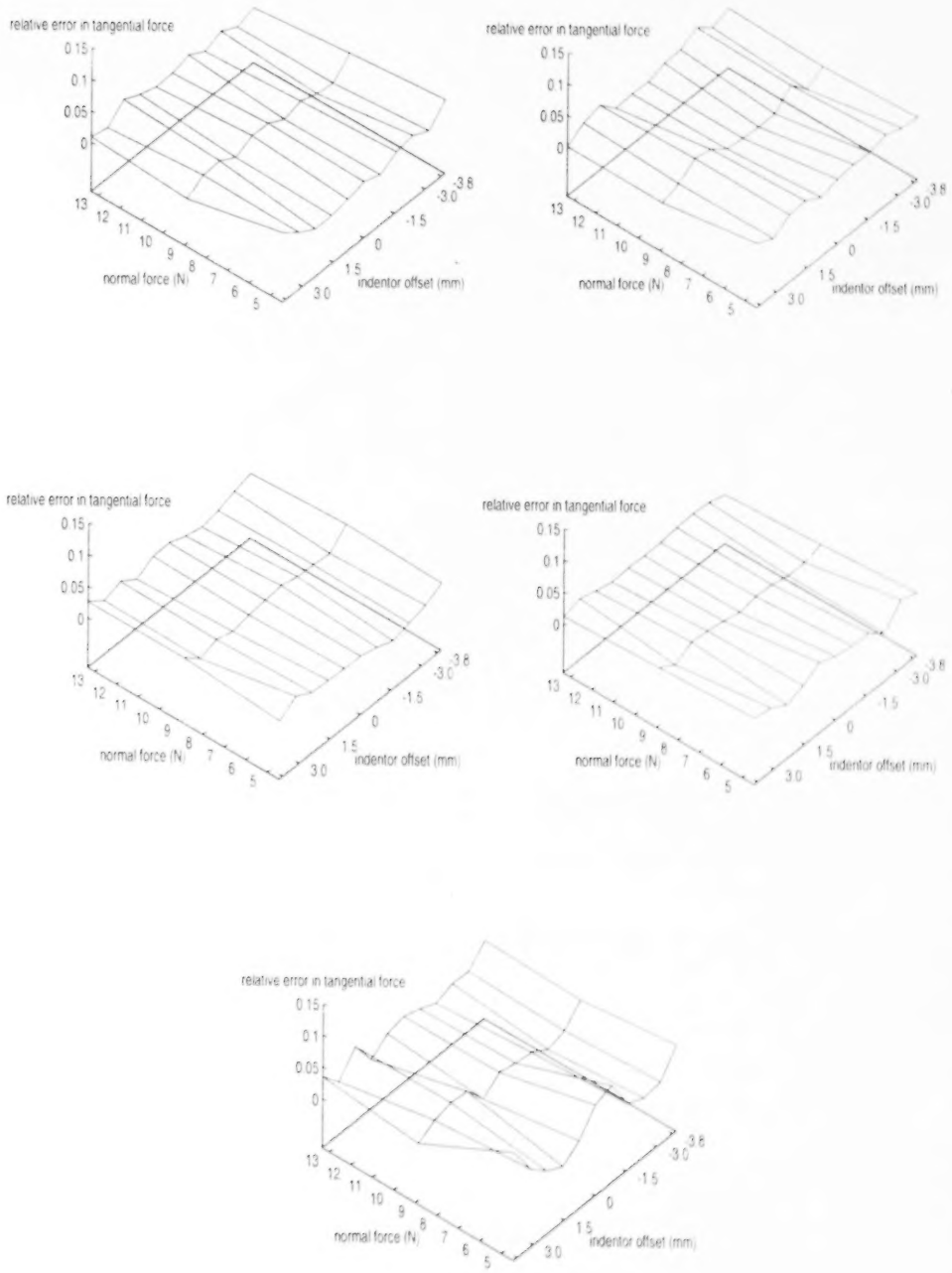


Figure 37: Relative error in tangential force. Each graph represents the error in tangential force recovery for a given applied normal force. The force values for this figure range evenly from 0.45 N in the top left corner to 2.2 N at the bottom.

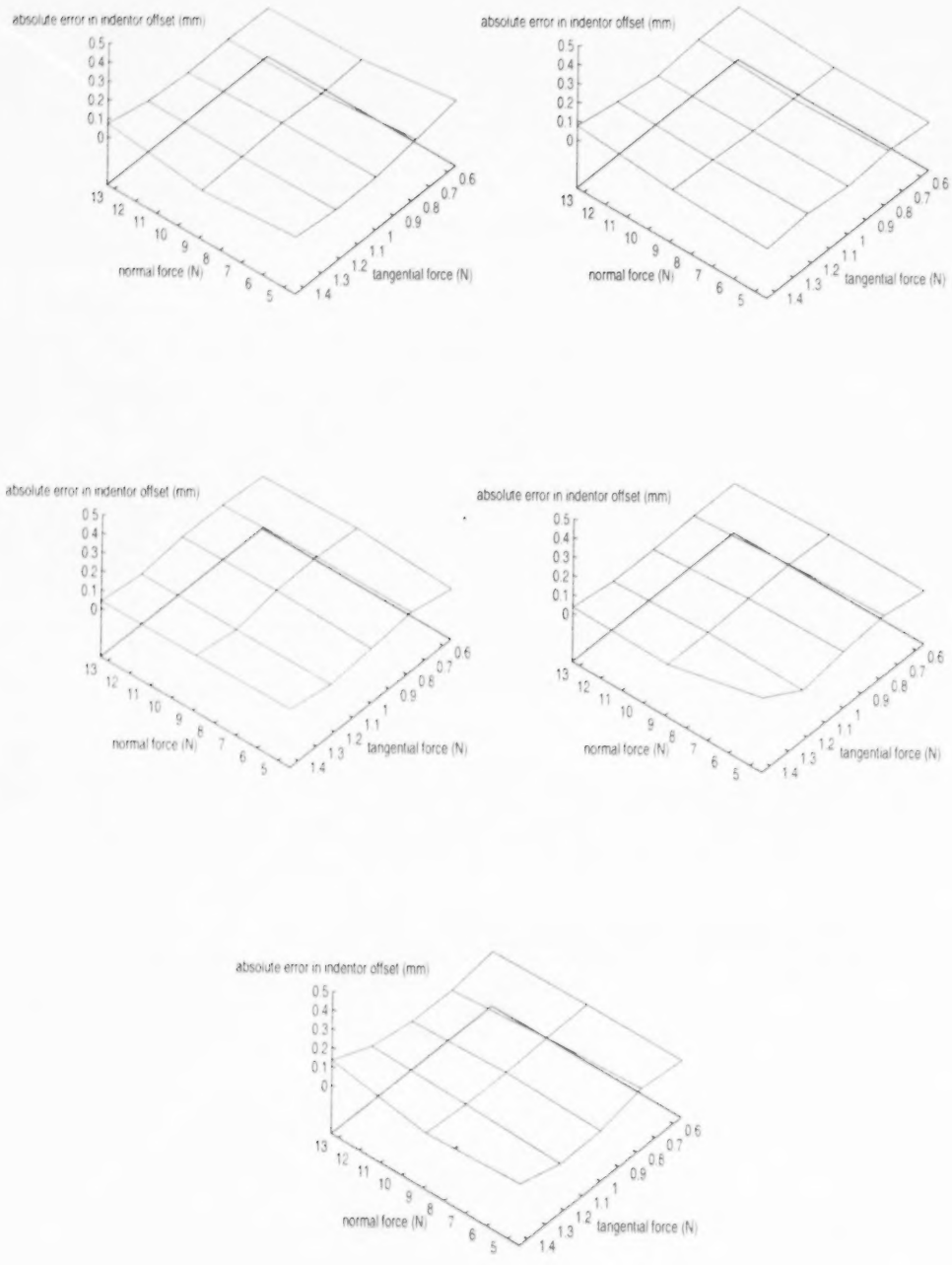


Figure 38: Absolute error in indentor position. Each graph represents the error in indentor position recovery for a given position. The positions (relative to the center of the sensor) for this figure range evenly from -4.0 mm in the top left corner to -0.8 mm at the bottom.

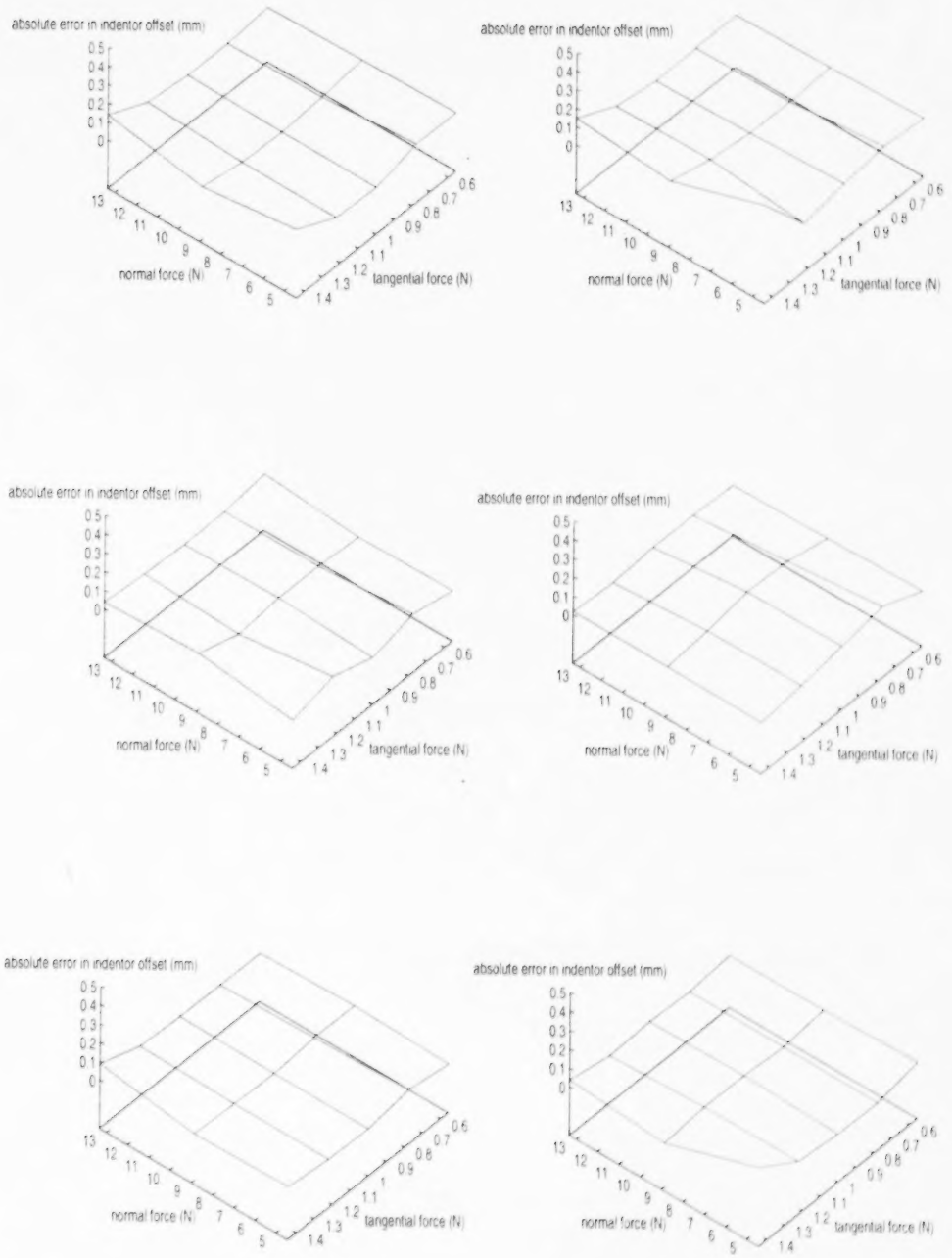


Figure 39: Absolute error in indentor position. Each graph represents the error in indentor position recovery for a given position. The positions (relative to the center of the sensor) for this figure range evenly from 0.0 mm in the top left corner to 4.0 mm at the bottom.

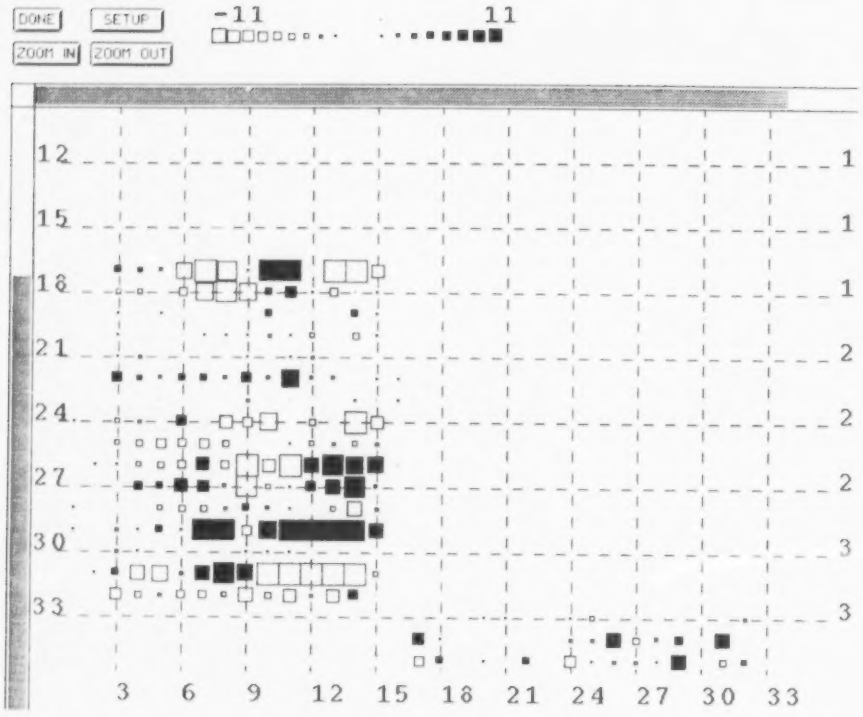


Figure 40: Hinton diagram of the trained weight values for the network recovering contact forces and indentor position.

Weight Analysis for the third experiment

The weight analysis for this experiment is more complicated than in the previous sections. Without the convenience of having a static position for the stress profiles, interpreting the weight diagram (Figure 40) is much more difficult and less sure.

The normal force output (node 33) depends very weakly on three nodes (20, 25 and 32) which perform the same kind of summing function seen in the previous experiments. The number of input nodes used in the sum is larger, most likely due to the fact that the stress profile may shift from side to side without a change in normal force. The tangential force output computation is less clear. The larger weighted factors in the output seem to roughly cancel each other out by having fairly equal and opposite weights for the input nodes. The result is a negative weight placed on

the inputs on the far left of the vector. This makes intuitive sense, as an increased tangential force tends to skew to profile to the right. The smaller weighted factor in this output has the opposite effect, giving a positive weight to the right side of the input vector. The overall effect is similar to that seen in the previous sections, but in this case the method does not work quite as well.

The indentor position recovery is again more complex than the tangential force case. Several of the hidden layer nodes have significant weights applied to them.

4.2 Experimental Results

In order to show the applicability of the simulated results to the physical problem, a simple experiment was set up where an *Interlink* force sensor was used to collect stress profiles corresponding to a set of different normal forces, an example of which is shown in Figure 41. A total of 19 different normal forces were applied to the sensor to form the training data set, and another 19 profiles were used to form the testing set. The network was trained for 5000 iterations before the testing set was applied to assess the performance.

As can be seen in Figure 42, the network was able to recover the applied normal force with very little error, despite the extremely small training set. Also, the sensor output shows that noise is present in the data, making the problem more complicated still. While far from an exhaustive test, this experiment shows that the methods being used in the previous experiments are applicable to the physical problem.

4.3 Summary of Results

The results presented in this chapter have shown that the neural networks were able to learn the required mapping functions with very little errors. A summary of the

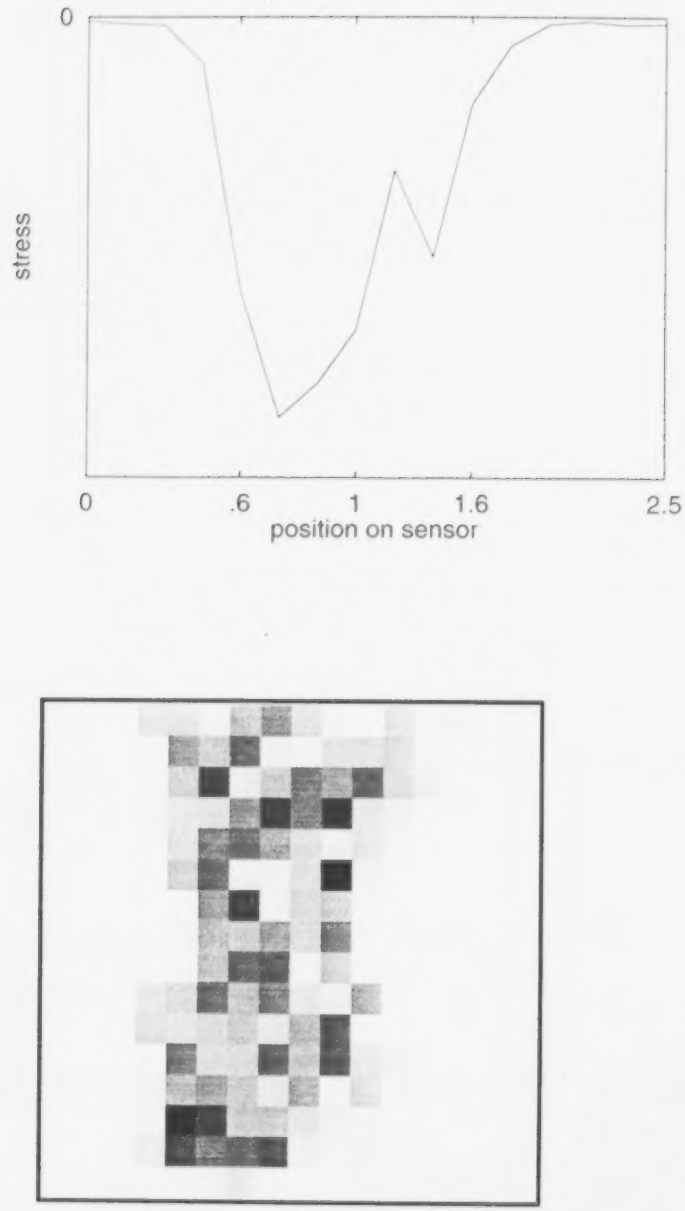


Figure 11: Sample stress profile from the training set used in the physical experiment. The normal force applied in this case was approximately 13 Newtons. The stress profile (top) was generated by averaging the rows of the actual sensor output (bottom).

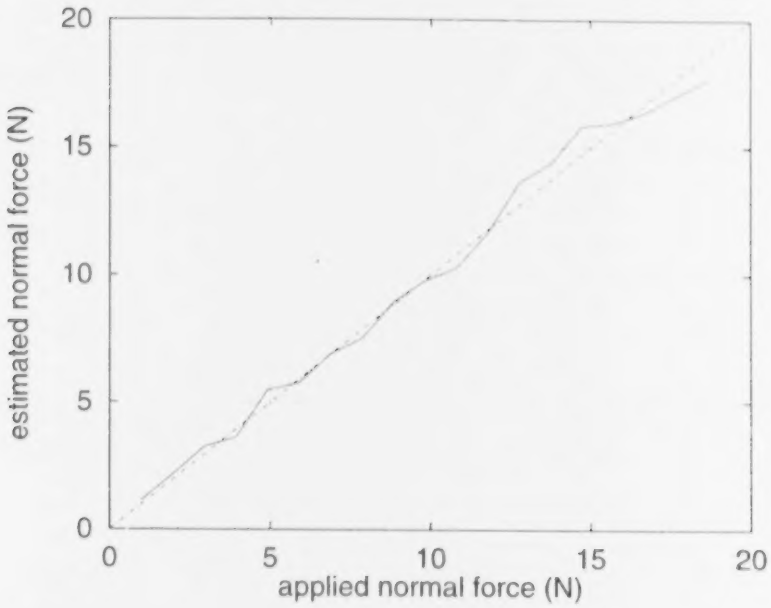


Figure 42: Normal force recovery for the physical sensor experiment.

Parameter	*Minimum*	*Average*	*Maximum*
Normal Force	0%	1.1%	14%
Tangential Force	0%	3.5%	18%

Table 4.4: Summary of errors for the network recovering normal and tangential forces only (first experiment)

network errors can be found in Tables 4.4, 4.5 and 4.6. In Table 4.6, the average change in indentor position was 2 mm, making the largest error approximately 10%.

Parameter	*Minimum*	*Average*	*Maximum*
Normal Force	0%	0.9%	12%
Tangential Force	0%	4.4%	19%
Indentor Width	0%	1.1%	15%

Table 4.5: Summary of errors for the network recovering applied forces and indentor width (second experiment)

Parameter	*Minimum*	*Average*	*Maximum*
Normal Force	0%	0.3%	1.4%
Tangential Force	0%	1.8%	9.4%
Indentor Position	0 mm	0.04 mm	0.2 mm

Table 4.6: Summary of errors for the network recovering applied forces and indentor position (third experiment)

Parameter	*Minimum*	*Average*	*Maximum*
Normal Force	0.3%	5.0%	15%

Table 4.7: Summary of errors for the network in the physical experiment

Chapter 5

Conclusions and Further Research

Tactile sensing is a very important part of robot manipulation. Force and position feedback about the object being handled makes fine manipulation tasks possible to accomplish. The tactile sensing technology available today does not directly give these high level contact parameters, thus extraction of these parameters is a critical part of manipulation.

Due to the complexity of a general forward sensing model, simplifying assumptions are often used to make a solution tractable. These assumptions do not hold in many cases, making the solutions less accurate than might be needed. Numerical methods, such as finite element analysis, can be used to avoid the use of simplifying assumptions. Unfortunately, numerical methods cannot be inverted analytically, making the recovery of contact parameters difficult. Neural networks can learn complex functions from example input/output vector pairs and therefore should be able to approximate the inverse function from a set of numerical data provided by a finite element model.

In the experiments presented in this thesis, backpropagation neural networks were used to learn the functions that map sensor stress profiles onto contact parameters. The stress profiles were generated using a finite element analysis model of a rectangular indentor on a thin elastic strip with sensing elements embedded at the bottom.

The recovery of four contact parameters was attempted:

- applied normal force
- applied tangential force
- indentor width
- indentor position

After relatively short training times, the network performed extremely well in all cases, with average relative errors being less than 5%. A physical force sensing array was used to produce stress profiles for differing normal forces. These profiles were used in training the same network used in the previous experiments, and the trained network was tested on stress profiles generated by physical sensors. The network performed well, showing the applicability of the previous simulated results to the physical problem.

5.1 Further Research

While the results presented in this thesis are very exciting, more work should be done to determine the range of contact parameters that can be recovered using neural networks. Recovery of information when using different indentor shapes would be essential to ensuring the usefulness of this approach in arbitrary manipulation tasks.

More experimentation with stress profiles generated by a physical sensor needs to be done to ensure that the model behaves correctly under all contact parameter combinations. Also, as noise inherent to any physical system can cause performance degradation, tests should be run to determine the networks performance on noisy data. The physical testing done in this thesis was very limited and was meant only to demonstrate the possibility of applying the model to real tactile sensors.

Appendix A

Finite Element Model

This appendix is reprinted with permission from [SZT93].

A.1 Material Properties

The elastic layer covering the sensor was assumed to be made of rubber. The material properties used for this elastic layer were taken from Ashby and Jones [AJ80]. The indenter was assumed to be made of steel and its material properties were taken from the library provided by the FEM software used for the analysis [ALG90]. These values are shown below.

	Young's Modulus E (dynes/cm^2)	Poisson's Ratio ν	Density ρ (gm/cm^3)
Rubber	2.758e+08	0.49	0.91
Steel	2.069e+12	0.30	7.85

Since it is assumed that the materials are linear and isotropic, the modulus of rigidity μ is determined from the Young's modulus E and Poisson's ratio ν using the equation below.

$$\mu = \frac{E}{2(1+\nu)} \tag{A.1}$$

A.2 FEA Model of Layer

Figure 43 shows the two-dimensional model used for analyzing the applicability of the half space assumption. The bottom surface of the layer is completely constrained while a line load is applied at the top surface. The layer is divided into a number of rectangular parts called elements. The corners are called nodes. These elements are referred to as two-dimensional stress elements and the accuracy of the solution is determined by the geometrical properties of these elements. The more the number of elements, the more the accuracy of the FEA solution. However, the more the number of elements, the more the computational and memory requirements of the FEA algorithm. The accuracy of the solution is also affected by the ratio of the two sides of an element. This ratio is called the aspect ratio and it should be as close to 1 as possible. Very high (10) or very low (0.1) aspect ratios can lead to inaccurate results. The number of elements per unit area is referred to as the mesh density. A number of different mesh densities were tried to select the mesh densities used in the analysis. The mesh density can be variable and it is usually made higher in regions where the stresses change more rapidly. In the three-dimensional model used to study the effect of the plane strain assumption, the elements are three-dimensional and are called brick elements or three-dimensional stress elements.

A.3 FEA Model of Contact

The indenter is also explicitly modeled using the FEM. A number of different mesh densities and indenter lengths were tried before arriving at the mesh densities used in the analysis presented in the paper. The indenter and the layer were modeled as two

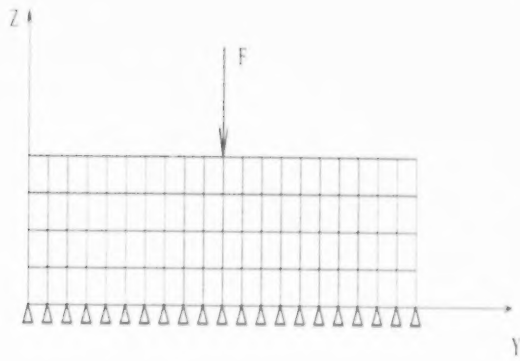


Figure 43: *A simplified model of the elastic layer used in the first part of the paper. The actual mesh density used in the analysis was higher than that shown in this figure.*

separate objects. The elastic layer was modeled as a mesh of 64x5 elements. The FEA software requires that all objects in the analysis be rigidly constrained. The elastic layer was rigidly constrained at the bottom surface. The indenter was constrained by using two elastic boundary elements at one corner and one elastic boundary element at the opposite corner. The elastic boundary elements act as springs and so allow some movement when force is applied. The contact between the indenter and the elastic layer was modeled using gap elements. These elements are used to connect the two surfaces in contact. One gap element connects one node on the indenter and one node on the elastic layer. These elements are inactive as long as the distance between the connected nodes is greater than zero. When the distance becomes less than zero, these elements apply opposite forces at the two connected nodes so that the gap distance between the two nodes remains zero. The applied force depends upon a user specified stiffness. The complete model is shown in Figure 44.

A.4 Modeling Friction

To analyze the effect of tangential forces on contact, a model of friction is required. We used a simple local model of friction [KO88]. We assumed that in the area of contact, the force due to friction at any point within the area of contact is proportional to

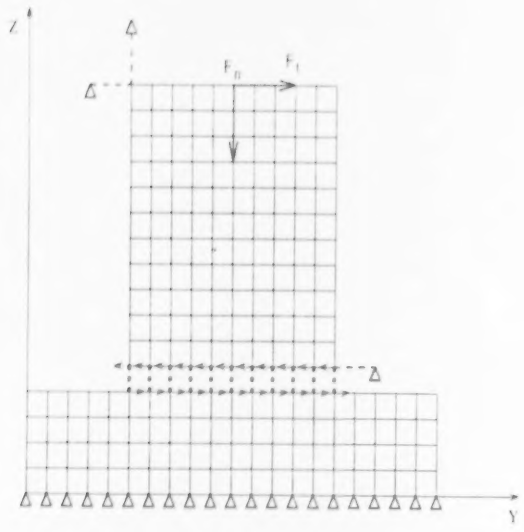


Figure 44: *A simplified model used for examining the relationship between the stress distribution and the applied normal force, tangential force, and the indenter width. In the analysis, the mesh density used was higher than that shown in this model.*

the normal force at that point. Since the system is in static equilibrium, the total force due to friction must equal the applied tangential force. The final solution is obtained in an iterative manner until the normal and tangential forces within the area of contact become stable. Figure 45 shows the difference between the stress distributions obtained by using the simple model of friction described above and by rigidly constraining motion in the tangential direction (the rigid constraint provides the tangential reaction force needed for static equilibrium).

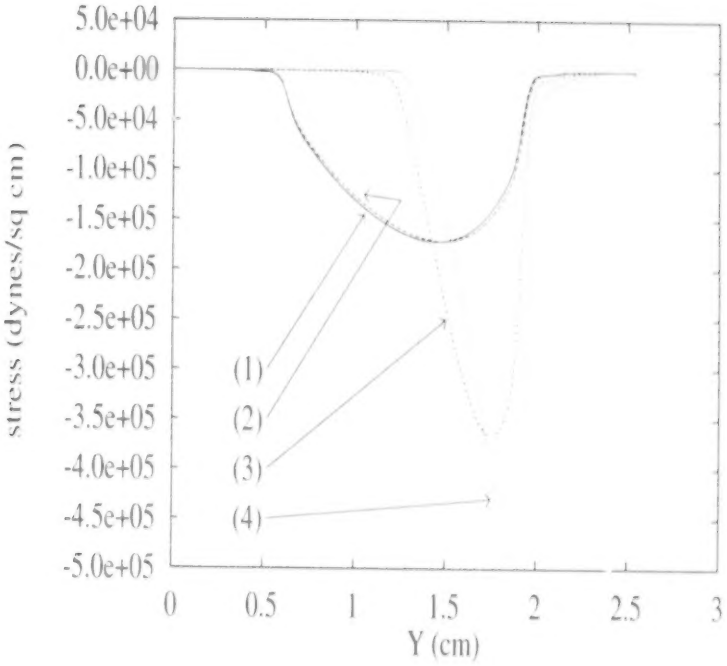


Figure 45: *The stress distributions with and without the friction model. All forces are measured in dynes/cm. (1) F_t = 2.2e4 without friction. (2) F_t = 2.2e4 with friction. (3) F_t = 1.5e5 without friction. (4) F_t = 1.5e5 with friction.*

Bibliography

[AJ80] M. F. Ashby and D. R. H. Jones. *Engineering Materials: An Introduction to their Properties and Applications*. Pergamon Press, 1980.

[AK69] J. B. Alblas and M. Kuipers. Contact problems of a rectangular block on an elastic layer of finite thickness, part i: The thin layer. *Acta Mechanica*, 8:133–145, 1969.

[ALG90] ALGOR Interactive Systems, Inc. *ViziCad Plus : Volume 2 Modeling for Finite Element Analysis*, 1990.

[BMP65] Raymond L. Bisplinghoff, James W. Marr, and Theodore H. H. Pian. *Statics of Deformable Solids*. Dover Publications, 1965.

[Bur87] David S. Burnett. *Finite Element Analysis : From Concepts to Applications*. Addison-Wesley Publishing Company, 1987.

[CMDR+92] G Canepa, M. Morabito, Danilo De Rossi, A Caiti, and T. Parisini. Shape from touch by a neural net. In *IEEE 1992 International Conference on Robotics and Automation*, pages 2075–2080, 1992.

[CZR94] N. Chen, H. Zhang, and R. Rink. An anlytical model for tactile sensing of point contact. Submitted to the 1994 IEEE International Conference on Robotics and Automation, San Diego, CA, 1994.

[DRCBC91] Danilo De Rossi, A Caiti, R Bianchi, and G Canepa. Fine-form tactile discrimination through inversion of data from a skin-like sensor. In *IEEE*

1991 International Conference on Robotics and Automation, pages 398–403, 1991.

[Fea92] Ronald S. Fearing. Basic linear elasticity. In Howard R. Nicholls, editor, *Advanced Tactile Sensing for Robotics*. World Scientific, 1992.

[Gro93] The SNNS Group. Stuttgart neural network simulator user manual : Version 3.1. Technical Report 3, Institute for Parallel and Distributed High Performance Systems, University of Stuttgart, Germany, 1993.

[HN89] Robert Hecht-Nielson. *Neurocomputing*. Addison-Wesley Publishing Company, Inc, 1989.

[Hop82] J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. In *Proceedings of the National Academy of Science*, pages 2554–2558, 1982.

[Int89] Interlink Electronics. *Product Design Guide*, 1989.

[KO88] N. Kikuchi and J. T. Oden. *Contact Problems in Elasticity : A Study of Variational Inequalities and Finite Element Methods*. Society for Industrial and Applied Mathematics, Philadelphia, 1988.

[NL89] H. R. Nicholls and M. H. Lee. A survey of robot tactile sensing technology. *International Journal of Robotics Research*, 8(1):3–30, 1989.

[Pav94] Pavan Sikka. *personal communication*, 1994.

[PKC*88] Y.C. Pati, P.S. Krishnaprasad, C.T.Yao, M.C. Peckerar R. Yang, and C.R.K. marrian. Neural network for tactile perception. In *Proceedings of the 1988 IEEE International Conference on Robotics and Automation*, pages 346–139, 1988.

[PKP92] Y.C. Pati, P.S. Krishnaprasad, and M.C. Peckerar. An analog neural network solution to the inverse problem of "early taction". *IEEE Transactions on Robotics and Automation*, 8:196–212, 1992.

[Ros58] Frank Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychology Review*, 65:386–408, 1958.

[SZT93] Pavan Sikka, Hong Zhang, and Roger W. Toogood. On modeling tactile sensors. Technical Report 8, Department of Computing Science, University of Alberta, 1993.

[TWR92] M. Thint, P.P. Wang, and J. Rebman. A study of back-error propagation networks for a trainable tactile pattern classifier. *Engineering Applications of Artificial Intelligence*, 5:205–214, 1992.

[WS92] A.J. Worth and R.R. Spencer. A neural network for tactile sensing: the hertzian contact problem. In *IEEE Transactions on Systems, Man, and Cybernetics*, volume 22, pages 177–182, 1992.